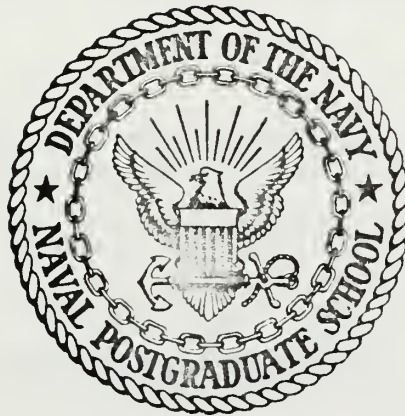


THE DESIGN, DEVELOPMENT AND TRANSLATION
OF GENERAL PURPOSE SOFTWARE FOR THE
P3C AIRCRAFT'S DIGITAL COMPUTER

Dennis Edward Ray

NAVAL POSTGRADUATE SCHOOL

Monterey, California



THESIS

THE DESIGN, DEVELOPMENT AND TRANSLATION
OF
GENERAL PURPOSE SOFTWARE
FOR THE
P3C AIRCRAFT'S DIGITAL COMPUTER

by

Dennis Edward Ray

Thesis Advisor:

G. H. Syms

December 1971

Approved for public release; distribution unlimited.

The Design, Development and Translation
of
General Purpose Software
for the
P3C Aircraft's Digital Computer

by

Dennis Edward Ray
Lieutenant, United States Navy
B.S., United States Naval Academy, 1964

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the
NAVAL POSTGRADUATE SCHOOL
December 1971

ABSTRACT

The Navy has incorporated a modified UNIVAC 1830-A (CP 901 or ASQ/114) "mini" digital computer into its P3C aircraft. This ASQ/114 computer system is presently used only during aircraft testing and flying. In the near future, fifty or more of these digital systems will be operational and will sit virtually idle about 40% of the time. Hence, this project was undertaken to improve the computer utilization, and to provide the individual squadrons with an administrative computer capability.

Six specific tasks leading to the implementation of a general purpose operating system have been undertaken: a feasibility study, development of a CP 901 translator, design and development of an assembler, design study of the bootstrapping technique, design of a FORTRAN compiler, and the design of a control operating system.

The documentation of these six tasks is intended to aid in the development of the final system.

TABLE OF CONTENTS

I.	INTRODUCTION -----	5
II.	OBJECTIVES OF THE RESEARCH -----	7
III.	ASQ/114 COMPUTER SYSTEM -----	9
	A. ASQ/114 HARDWARE -----	9
	B. ASQ/114 SOFTWARE OPERATING SYSTEM -----	14
IV.	FEASIBILITY STUDY, CRITERIA AND RESULTS -----	16
V.	DESIGN, DEVELOPMENT AND DOCUMENTATION OF THE CP 901 TRANSLATOR ON IBM 360/67 -----	21
VI.	DESIGN, DEVELOPMENT AND DOCUMENTATION OF THE CP 901 ASSEMBLER WRITTEN ON IBM 360/67 -----	23
VII.	BOOTSTRAPPING TECHNIQUE USED TO DEVELOP CP 901 ASSEMBLER -----	28
VIII.	DESIGN CRITERIA FOR A FORTRAN COMPILER FOR THE CP 901 -----	30
IX.	DESIGN CRITERIA FOR THE CONTROL OPERATING SYSTEM FOR THE CP 901 -----	35
X.	CONCLUSIONS AND RECOMMENDATIONS -----	38
	APPENDIX A. INSTRUCTION WORD FORMATS -----	40
	APPENDIX B. BOOTSTRAPPING GENERATION SCHEME -----	41
	APPENDIX C. BNF ASSEMBLER SYNTAX -----	43
	APPENDIX D. BNF FORTRAN COMPILER SYNTAX -----	44
	CP 901 TRANSLATOR LISTING -----	48
	CP 901 ASSEMBLER LISTING -----	76
	BIBLIOGRAPHY -----	88
	INITIAL DISTRIBUTION LIST -----	89
	FORM DD 1473 -----	90

LIST OF TABLES

TABLE I. Definition, Functions and the Number of Machine Code Translations	----- 25
---	----------

I. INTRODUCTION

The ASQ/114 special purpose digital computer system is an integral, perhaps the most important, part of the Navy's P3C Orion ASW aircraft. This aircraft is in fact a computerized integration of crew operations, aircraft weapons and support systems, sensors, and automatic navigation and steering. This integration of airborne sensors with a digital computer is termed the A-NEW concept. The major advantage of the A-NEW system lies in the computer's programmed ability to accomplish the myriads of detailed tasks in order to make logical decisions faster and more accurately than can be done manually.

This system incorporates computer-controlled displays at the TACCO (TACTical CoOrdination Officer), the Pilot and the non-acoustic operator stations for the presentation of aircraft, sonobuoy, and visual contact information. These displays are continuously updated, and with the aid of a relatively simple set of function switches, each operator controls and modifies his display to satisfy his station requirements. Additional displays at the TACCO and the NAV/COM operator stations enable them to view and modify numerous tables containing data regarding the status of equipment, weapons, search store devices, audio systems, and navigation. The computer has released the Navigator of virtually all of the mental integration and hand calculation normally required in the solving of the navigation problems. It regularly receives navigational data to update the aircraft displays; but also continuously monitors the active navigation system operation and automatically selects an alternate system in the event of a failure or malfunction. The computer can display upon request either a latitude/longitude or a grid co-ordinate position of

the aircraft, sonobuoys, contacts, or other selected positions. Thus by specifying preselected positions, the computer can provide precise steering courses to that selected point. The computer further manages the stockpile of weapons and search devices, etc., on board and automatically presents TACCO with alternatives for improper or unfulfillable selections.

The system as installed in the P3C Orion provides for the automatic recording of essential mission data on magnetic tape. This dayfile is taken as snapshot views of the mission profile at preselected regular intervals in the flight or whenever a significant event occurs.

A section of the ASQ/114's central processor, namely the bootstrap memory, is composed of 512 words of non-alterable code for use in loading the system's operating program and also in the automatic restart of the computer. This bootstrap memory is electrically unalterable and provides the restoration capability of the computer to save its environment, the entire aircraft status, and the display system. Thus minor computer interruptions cause only a temporary delay in the system operation, as a virtually instantaneous restore function can be executed which automatically reloads the operating system and the last status of the aircraft allowing normal system operation to continue without loss of information.

The heart of the A-NEW system is then the CP 901 digital computer which, under the control of the operational program, frees the aircraft operators from manually performing a number of functions. This in turn allows the operators considerably more time to concentrate on and perform their primary mission task (ASW problem).

II. RESEARCH OBJECTIVES

The primary goal of this research was based upon the satisfactory completion of a feasibility study of utilizing the special purpose digital computer system (ASQ/114) as a general purpose system (section IV). In essence then the main goal was to provide the ASQ/114 system users with the software to utilize the hardware in a general purpose digital system and, thus improve upon the system utilization. System utilization is a computer factor defined as the ratio of a system's actual processing use time to the possible time. For the ASQ/114 system this factor was conservatively estimated as 60%. There were many sub-goals categorized during the initial description of the problem. Of these, five have been completed; the description of which make up a substantial part of this report. These are described in sections V through IX.

The non-availability of an ASQ/114 system in the local area required the development of the CP 901 translator on the NPS IBM 360/67 (section V). Its primary purpose is to provide a testing tool for programs coded in UNIVAC machine code. The original CP 901 assembler (section VI) was programmed in IBM PL/I. It also is a tool, but is designed to be used in connection with the bootstrapping technique (section VII) to produce a CP 901 assembler written in CP 901 assembly language. This assembler can in turn be used to produce a FORTRAN compiler (section VIII) and a control operating program (section IX) as described in section VII.

By completing these five sub-goals, an individual or group associated with NPS could readily complete the installation of an ASQ/114 general operating system with a minimum of effort. Sufficient documentation and

description of these goals is provided in order to permit followup efforts to continue from this point, rather than requiring additional effort be expended in investigating system factors already considered.

III. ASQ/114 COMPUTER SYSTEM

A. ASQ/114 HARDWARE

The ASQ/114 system hardware was developed by the UNIVAC Corporation as a specially configured 65K memory UNIVAC 1830-A digital computer. The 1830-A is a miniaturized (8 cubic feet, 395 lbs.) general purpose, real-time, stored program machine that is composed of four relatively light weight modular assemblies: central processor unit, memory unit, power supply unit, and input/output unit. The CPU is an example of a typical processor configuration with a control section and an arithmetic section which together perform arithmetic and logic functions under the direction of the control section. The memory unit consists of ferrite core modular arrays of 4096 thirty bit words. Memory is overlapped between 16K word banks, with a maximum machine memory capacity of 128K. The overlap operation involves addressing any two memory units simultaneously. The power supply unit converts the input voltage, 115 VAC 400 cycles to regulated direct current. The input/output unit provides up to sixteen thirty bit parallel input or output channels.

The military modification to the 1830-A occurred primarily in the sixteen channel and interface units required for the specialized aircraft peripheral I/O units. The interfacing of all peripheral equipment is controlled in either one of the three logic units (LU-1, LU-2, and LU-3) or one of the five converter or interface units (teletype interface, DIFAR interface, synchro-digital interface connector, radar interface, and a data bank converter).

The major peripheral equipment includes two seven track magnetic tape drives, a teletype CRT/teletype display as well as various minor output

lights, panels, and CRT displays. Normal military input units include radar, navigation, doppler, hand keysets, control panel settings, various acceptance switches, and other special military input devices.

Operation in the central processor utilizes overlap memory techniques to decrease the effective instruction set execution times. In the ASQ/114 overlap is the simultaneous reading of the instruction for the next operation from one memory unit and the operand for the present instruction from another memory unit. The CPU also provides the I/O capability of overlapping memory addressing. In this case overlapping is the ability for the I/O and the program instruction set to simultaneously access different memory units. The I/O operation has priority in the case of all conflicts.

Within the CPU, the control section utilizes the U register, a thirty bit register, to hold the instruction word during execution of an operation. The function code and the various instruction designators are translated from the appropriate bits of this register. If an address modification is required before execution, the contents of the appropriate B register is added to the proper section of the U register. The B register set is a group of eight fifteen bit registers used as address modification registers or as index registers.

The P register is a fifteen bit register that holds the memory address of a computer instruction word, that of the next instruction to be loaded into the U register for execution.

The K register functions as a shift counter for all shift operations. Such operations that use this register are the arithmetic functions of multiply, divide, or square root.

Associated with the A register, the conventional accumulator, are the CPU working registers the A* and the X. A typical function of addition

would shift the contents of the A register into the A* register. The X and the A* registers would then be added in a parallel bit operation with the results being placed into the A register.

The thirty bit Q register is principally used during the multiply and divide operations. The contents of both the A and the Q register may be shifted left or right or may be combined into one sixty bit word. All shifts are accomplished by the parallel exchange network in conjunction with the A* and Q* registers. Neither the A*, Q*, nor the X register is directly addressable, as these are only used by the CPU for the exchange of data within the arithmetic section.

Each 16K memory unit contains two registers, the S and the Z. The S register contains the fourteen bit address of the location within the unit being referenced. The Z register holds the thirty bits of data being written into or read from the memory location specified by the S register.

In order to access all four memory units, it is necessary to use the "indirect" memory addressing scheme. The normal address portion of an instruction in the U register is fifteen bits, which permits access to only 32K of memory (2 to the 15th power is 32K). Thus in the direct mode of addressing bit fifteen of the address selects either unit zero or unit one and the remaining fourteen bits are passed to the selected unit's S register. In the "indirect" mode of addressing bits eleven through fifteen of the instruction's address select one of the sixteen six bit Absolute Page Registers (APR). The six bits of the selected APR are transferred to the R2 register. The upper three bits determine the memory unit to be used and the lower three bits determine the page within the unit to be addressed (pagesize is 2K). The original remaining eleven bits of the address then specify the word within the selected page.

All input/output operations are passed through the computer's I/O section, which provides for twelve input channels and sixteen output channels. These channels are assigned priority in descending order from fifteen to zero. They are further divided into four 4 channel groups with descending priority from three to zero. Only one channel in a group may be active at any time, but all four may be transmitting data simultaneously.

There are four basic methods of communication using the channels between the computer and any of the peripheral equipment or devices. Data transfers may be made with or without Monitor. With Monitor an interrupt is generated upon completion of the data transfer, so that immediate processing of the change of data can be initiated. Without Monitor the data transfer is merely completed and no specific action is taken until a program instruction is encountered directing that action. The four methods of data transfer are:

1. Input data transfer (with or without Monitor) is the normal method of transferring data from a peripheral unit to the computer's memory.
2. Output data transfer (with or without Monitor) is the normal method of transferring data from the computer's memory to a peripheral unit.
3. External function (with or without Monitor) is an output function of higher priority than a normal output transfer, and usually requires the peripheral unit to perform some action or provide some reply. External function with force is a variation of this mode that passes some form of data to a peripheral unit that is not capable of requesting the data.

4. External Interrupt is the highest priority of input data transfer and is usually used to report a significant change of status.

All data transfers are accomplished via a buffering process. This buffering allocates an area of memory into which or from which the data transfer is made. It then permits normal program execution and data transfer to proceed on a time shared basis until all of the words have been transferred.

Interrupts (internal and external) temporarily halt the executing instruction and place the attention of the CPU upon the event that generated the interrupt. Internal interrupts stem from computer power failures, program faults, memory protection violations, and I/O interrupts generated upon the completion of data transfers with Monitor. External interrupts include those generated from logic unit power failures, input contacts, weapon release switch activations, etc. All interrupts cause a jump to a memory location containing the instructions for handling the specific type of interrupt that was generated.

The inputs and outputs to and from the computer's memory consist of the data transfers and coded control signals. These control signals are specific bits of information to indicate requests or acknowledgments of the receipt of some transfer of information. Data transfers are thirty bit words transmitted over thirty parallel data lines which may carry normal thirty bit data words or coded signals. There are twelve input data points, five of which are multiplexers. Similarly, there are sixteen output stations, six of which are multiplexers. For each station there exists a special unique format word identifying the station and desired function of operation. Within these thirty bit words there exist numerous field variables dependent upon the particular station involved.

B. ASQ/114 SOFTWARE OPERATING SYSTEM

The operating program of the ASQ/114 system is a 62K program that is a pseudo simulation of the environment of the aircraft. All of the aircraft's variables are actively retained in core memory at all times. These variables are updated as necessary and on a continuing basis in order to maintain the aircraft status and system displays. At the same time the system keeps a copy of the memory on magnetic tape. This tape copy is used to recover the system and the environment in the event of a temporary computer or system malfunction. Magnetic tape transport #1 is thus dedicated to the operating system and the recovery program. On the other hand magnetic tape transport #2 is dedicated to receive the running picture of the snapshots of the events occurring during the flight. This dayfile is saved for evaluation at a later date.

The system responds to each input request, first determining the unit from which the request was made. Then it ascertains the word format of the request and breaks down the coded fields in order to respond to the request. Once the proper response has been computed, it is coded into thirty bit words of the proper format for the requesting station. The information is then transferred to the requesting peripheral unit and there it is decoded into the proper display format in order to be displayed.

The operating program is a program written in the seventy-two available basic operation codes (3652 operations with the various modifiers). These fall into the categories of arithmetic operations of add, subtract, etc., as well as jumps, shifts, repeats, comparisons, load and store functions, various I/O functions and interrupts. The basic operations codes (opcodes) are combined with the various modifiers and the

operand address into one of five possible format types. These formats are illustrated in Appendix A. The formats are variations of the bit pattern interpretations used to identify the different functional instructions and the included data of the format word. These types include two variations of a general instruction format, a special I/O format, a special non-arithmetic format, and an addressing format. These instruction formats are broken down in the CPU and its associated registers to perform the desired function.

Within the framework of the operating program numerous instruction sets have been built to perform the wide variety of complex operations required. These operations include a communications network, an external computer linking capability, the analog input linking and the thousands of record keeping tasks maintained within the system. All of these factors provide an operating aircraft with a capability superior in every way to previous versions of the aircraft, combined with a significantly improved safety factor for all of the aircraft system operators.

IV. FEASIBILITY STUDY: ASQ/114 AS A GENERAL PURPOSE SYSTEM

In evaluating the ASQ/114 system for its general purpose capabilities, knowing its special purpose application, eight specific areas were considered. These eight categories fall under the following general headings:

1. History of the system's digital components
2. Untried procedure problem areas
3. Implementation problem areas
4. Anticipated benefits
5. Operating overhead factors
6. Personnel problems
7. Digital system language
8. System design concepts

The heart of the ASQ/114 system is a modified UNIVAC 1830-A digital computer. The basic design of the 1830-A is from a long line of production models of various UNIVAC miniaturized computing systems. The immediate predecessor of the 1830-A was the Phoenix system which used a 1830 as its digital computer. Thus the digital components of the system are totally compatible with the standard UNIVAC system. The seventy-two basic machine instructions are identical with those for the standard 1830-A computer. Similarly the two magnetic tape drives of the 1830-A system are machine compatible with most UNIVAC systems as well as a few IBM systems. ✓

In the initial testing and evaluation of the A-NEW system, a set of diagnostic programs was developed by the UNIVAC Corporation to isolate equipment errors in a minimum of time. The existence of this set of routines lends credibility in believing that a general programmer can in fact program the A-NEW system for general purpose use.

There are many pitfalls in attempting to install a new operating system, especially where a present system does not exist. In this case there are no blocks to build upon. At best the system designer is attempting to fit a software package into an existing hardware system. Although difficult, this concept is certainly a reasonable, if not the only, approach to use in the development of an operating system. Ideally the design of an operating system should be developed simultaneously with the associated hardware, but in this case the software will at best be an adaptation. As such the efficiency of the operating system will be somewhat less than that of a simultaneously developed system. However, total efficiency of the system is not of significant concern providing reasonable program execution can be achieved, especially since the intended applications would utilize normally unused computer time.

The number of P3C aircraft equipped with ASQ/114 systems in the next two years is anticipated to be about fifty. Considering that as a minimum these systems will be idle about 40% of the time and as the number of aircraft systems increases, the percentage of idleness will undoubtedly increase (due to the increased aircraft maintenance and repair time). The total weekly computer time available then is about 3500 CPU hours — a considerable amount of potential compute power.

As the number of aircraft increases, likewise the number of records, files and supply functions at a Naval Air Station will also increase but at a rapid pace. The number of administrative personnel required to manage these areas will also spiral upwards. If the idle CPU time could be channeled towards managing this area, the end result would be a more efficient flight squadron.

Assuming that an operating system (control program, higher level language compiler, and an I/O monitor) could be developed, the benefits gained by using the computer's idle time must be weighed against the operating overhead. Since the system is composed of a miniaturized computer, the operating power requirements are very minimal, less than one kilowatt. Power can be provided either directly from the aircraft's auxiliary power supply or from an external cable source readily available in the normal aircraft support facility.

The most serious hardware limitation to the use of the ASQ/114 system as a general purpose digital system is the output facilities. The aircraft's line printer is a non-impact type that uses a heat sensitive type of paper. This paper costs about eighty cents for a page containing about twenty-five lines. The cost of a single line of about 100 characters is about 3.2¢ (about 160 times higher than normal computer printer output). The TACCO's CRT could as well be used for displayed output, but it is temporary and limited by the size of the screen. The approximate number of output lines visible at one time is eighteen. The only other feasible output device for general purpose use would be either of the system's tape drives. These tape drives are UNIVAC and IBM compatible. The drawback with these devices is that some other compatible computer system must be available for reading of the taped output generated from the ASQ/114 general purpose system. This limitation is discussed further below.

Of significant importance to the installation of a general purpose operating system is the present lack of EDP trained personnel within the aviation community, especially in the field of the management of data processing equipment. In order to adequately use the general purpose

system, it is imperative that knowledgeable personnel operate the system. It is anticipated that the minimum training required for an operator would be some background in programming coupled with three weeks of direct active training with the general purpose operating system.

In order for the system to yield the maximum usefulness throughout the range of locations of the ASQ/114 system, it was felt that since FORTRAN was probably the best known language, it should be the language implemented into the general purpose operating system. The FORTRAN compiler will be developed using the translator and the assembler (sections V and VI) along with the bootstrapping technique (section VII). The implementation programs will consist of a control program, a CP 901 assembler, and a FORTRAN compiler all written in CP 901 assembly language.

In summary then the greatest limiting factor in implementing a general purpose operating system is the limited I/O capabilities of the ASQ/114 system. The existence of the two tape drives provides the ease of handling file manipulation problems. The lack of significant output printing capability severely limits the size of a problem's printed output. Although the hardware output capability is limited, this does not preclude the use of the ASQ/114 system as a general purpose computer, especially for the many applications requiring a small amount of output. (If the system proves to be sufficiently useful and effective, then the possibility of acquiring and interfacing of a special output printer should be investigated). Also of the three possible input methods available in the present configuration (direct panel coding, keyboard typing, and magnetic tape) none are very satisfactory for long program inputs. Direct panel coding requires programming in the system's machine code which is a very long and tedious process. Keyboard typing appears to be

the most likely method of input to the system, whether for data input or for programming. Tape input would require an original tape generating program either on this system or on a compatible system. Another limitation of the hardware is the size of core memory (65K). The maximum size of any program is thus limited to something less than 65K. With judicious use of the control program to monitor the printed output as well as due concern for the limited size of a program, one can still accomplish a very wide range of computing problems. There is under development a 128K drum addition to the ASQ/114 system. The addition of this paging drum will more than double the possible program size and thus will eliminate size as a limitation of the general purpose system.

After considering the various factors, it appears that only the I/O problem is of significant concern in implementing a general purpose. By carefully programming around this limitation, many different utilizations of the ASQ/114's system idle time are feasible.

V. CP 901 TRANSLATOR

The CP 901 translator is a design and testing tool used in the development of the initial assembler. It is written in PL/I to be run on the IBM 360/67 at NPS, and is used to verify the machine code generated by the assembler and the compiler. Thus the assembler, the FORTRAN compiler, and all programs written for this system could be written and debugged at this installation, instead of requiring an ASQ/114 system from a P3C aircraft.

The translator is a direct coding of the repertoire of basic CP 901 instructions given in appendix A to reference 1. Of particular concern were the many variations affecting the basic seventy-two operation codes. Where possible and practical, these variations were coded as separate sub-procedures. Since the maximum allowable size of an array on the IBM 360/67 is 32K, it was necessary to split the dummy memory into an upper and a lower half. Considerable difficulty was encountered during all arithmetic operations due to the octal vs. hexadecimal representation in the CP 901 vs. the IBM 360. Thus it was necessary to force the conversion of all numeric information and addresses into octal representation prior to any arithmetic operation.

The translator program begins execution with memory address 620, the initial program loading point of the CPU's bootstrap loader. The dummy U register is set to the contents of memory address 620. The various fields are then isolated and a jump to the designated "opcode" is executed. Within each of the opcode sections the function of the opcode is performed as modified by the various designators. After completion of the function, the next sequential instruction word is loaded into the

dummy U register so that the same process can then be repeated. The sequential stepping through memory continues until either a programmed jump to another address is encountered or a termination occurs. All output from the executed routine is placed on output file sysprint on the IBM 360.

The internal coding of the functions within the various opcodes is reasonably straight forward. The comments that precede each section explain the function of the section as well as the variables and function calls. Wherever possible the use of PL/I built-in functions were utilized. The major functions so used were BOOL, BIT, BINARY, SUBSTR. All other functions or procedure calls are user-defined.

A listing of this translation routine is included after appendix D.

VI. DESIGN AND DEVELOPMENT OF IBM 360 CP 901 ASSEMBLER

One must always keep in mind that a computer only understands its own machine language as specified by the manufacturer. Programming in machine language is generally extremely laborious, because the language is usually very complex, unnatural from English, difficult to memorize, and very vulnerable to errors. These difficulties are emphatically true with the UNIVAC 1830-A machine instruction set. Although there are only seventy-two basic opcodes, the six modifiers actually provide for a total of 3652 machine instructions. Direct machine language programming is thus extremely tedious.

A basic assembly language is the next higher level of computer language. It more closely resembles English forms of expression, but is not easily recognizable as English due to the coding schemes utilized for each expression. Assembly language coding is significantly better for the programmer than machine coding; but often the programmer must write nearly as many assembly instructions as he did machine language instructions. Remember that the computer cannot execute instructions written in assembly language directly, but by means of the assembler translator, these statements are transformed into executable machine code. However, writing anything but very small programs in assembly language is still a long and tedious task. A means of reducing this task still further will be discussed in section VIII, but it requires an operational assembler for development.

To initially alleviate some of the problems of writing programs in machine language for the CP 901, an assembly language set was defined. This set is based upon an existing language and is expressed in the

Backus Naur Form (BNF) syntax and is given in appendix C. The definitions and the functions of the language statements are listed in Table I. Note that this general set does not conform directly with the machine language set, thus many assembly instructions will require multiple machine instructions. In this sense this assembler is a "macro assembler" rather than a one-for-one assembly to machine code translator.

This assembler written in PL/I contains significantly more operators than required for the initial generation of a CP 901 assembler. These excess operators have been provided as a direct guideline for future implementation using the bootstrap technique. Some of the operators associated with arrays and procedure calls are not defined or documented, as such they are intended for reference only.

There are thirty-two labels permitted in the assembler. These are numbered sequentially from 540 - 577 octal. All program object machine code will be loaded into memory starting at address 620 octal. With this fact in mind the address of the machine instruction labelled in assembly code is stored in memory at the address of the label itself. Note that the labelled address must be stored prior to a programmed reference to that label. This can either be a forced programmed storage or may be accomplished by the assembler. To assist the assembly language programmer, the number of machine instructions issued from every assembly instruction is listed in Table I. The programmer can thus determine the memory location of pertinent machine instructions for referencing and modifications. This is accomplished by using address 620 as the base and adding to it the number of machine instructions for each assembly instruction issued. Thus for a simple program set of ADD, ADD, MUL, DIV, the first machine instruction of the divide assembly instruction would be located in memory at address 625.

TABLE I

Operation, #1 , #2	Function	#Inst
ADD , X	Add to register A the value of field X	1
SUB , X	Subtract from register A the value of field X	1
MUL , X	Multiply register A by the value X	3
DIV , X	Divide register A by the value X, dividend in A, remainder in Q register	4
EXP , X	Raise A to X power, result in A	4
LES , X	Skip the next instruction if X is less than A	3
EQL , X	Skip the next instruction if X is equal to A	3
LEQ , X	Skip the next instruction if X is less than or equal to A	1
NEQ , X	Skip the next instruction if X is not equal to A	3
GEQ , X	Skip the next instruction if X is greater than or equal to A	3
GTR , X	Skip the next instruction if X is greater than A	1
NEG , X	Skip the next instruction if X is negative	3
NOT	Compliment A	1
AND , X	If X = A Compliment Q	5
BOR , X	If X or Q = 0 Compliment A	5
ONE , X	A = X Q = 0	1
TWO , X , Y	A = X Q = Y	2
LOD , X	A = X	1
STO , X	Store A in memory (X)	1
STD , X	Store A in memory (X) set A = 0	2
XIT	Restart the program, reset all registers	4
BFN , X	Branch forward X instructions	3
BFC , X	Branch forward X instructions if A greater than zero	3
BBC , X	Branch backward X instructions if A greater than zero	3

TABLE I (Continued)

Operation, #1 , #2	Function	#Inst
BRS , X	Branch to instruction address stored in memory address (X)	3
BSC , X	Branch to instruction address stored in memory address (X) if A \neq 0	3
NOP	Remain idle and wait for data channel	1
PRO	Procedure call (for future use)	
RTN	Return from a procedure (for future use)	
GET , X	Allocate X memory locations for an array (for future use)	
RET , X	Return saved area X (future use)	
RDV , X	Read variable into A using data channel X	3
WRV	Output A	3
WRS , X	Write string of X words stored at memory location 5 (11 words max.)	X+3
DMP , X	Output bits of A (future use)	
TAB	Future array use	
ROW	Future array use	
SCR	Future array use	
SAV	Future procedure use	
UNS	Future procedure use	
SUP , X	Future error handling call to the operating system	
LIT , X	A = X Q = X	1
EOF	Terminate execution set P = 0	1
DEL	Set A = 0	1
DUP	Set Q = A	1
XCH	Exchange A and Q	1

All of the comparison operators (LES, LEQ, GTR, etc.) require a branching instruction as the next instruction for the "false" path of execution. Note that all branching, whether forward or backward implies a knowledge of the actual machine instruction address. However, this indirect type of addressing is translated into direct instruction addressing via manipulation of the P register and the count of the number of machine instructions per assembly instruction.

Use of this assembly language implies a knowledge of the associated machine code. However, by carefully studying the assembly language operator descriptions, a competent programmer can write routines in CP 901 assembly language with a minimum of effort. Thus with this PL/1 version of the assembler at NPS, any program coding is possible on the school's IBM 360 computer. The object code produced would then be executable on the CP 901. In general very long assembly programs tend to be infeasible due to the tedious task of inputting this larger number of statements into the computer (as discussed above). In this case of a large program it would be necessary to use a higher level language translator to produce CP 901 machine coded programs.

VII. BOOTSTRAPPING TECHNIQUE

With the simple CP 901 assembler written in IBM 360 PL/1 and executing on the NPS IBM 360/67, the process of developing a complete CP 901 assembler written in CP 901 machine language through a series of executions is termed "bootstrapping".

One of the major concerns in the bootstrap development process is the absolute error free initial generation run. In order to facilitate this requirement, usually the initial generation is kept as short as possible in order to reduce the probability of logic and programming errors. This generation must also accept at least the minimum number of input instructions in order to generate a workable set of primary machine coded instructions.

Once a working assembler is generated in the primary machine language, it can be isolated as the operating assembler. This program can then be loaded into either the translation memory or the actual CP 901 system for execution. Under this execution the second source input program produces the second generation CP 901 assembler.

Consider the generation scheme as depicted in appendix B. The initial assembler generation source input is a series of instructions acceptable to the PL/1 CP 901 assembler (CP ASS to CP 901 TRANS). The output from this execution is a simple assembler written in CP 901 machine language. This coded set of instructions will then accept CP 901 assembly language programs under execution on the CP 901 system. By making incremental changes to the source program and executing the change on the previous assembler, the size of the next version of the assembler can be increased.

Although this description is a rather oversimplified version of the actual bootstrapping process, it does illustrate the principles involved. The necessary programming to achieve the initial assembler is quite substantial. Once the initial assembler has been generated, however, the step by step additions become relatively easy to install. More effort, however, is required in the testing of new additions than in the installation of the change itself.

VIII. DESIGN CRITERIA FOR A FORTRAN COMPILER

The standard electronic computer is not an electronic brain, but a simple machine that does calculations on data by obeying a sequence of instructions. This series of instructions is uniquely peculiar to each machine, its own machine language. As is amply documented, programming in machine language for anything but the most trivial of tasks is beyond the general programmer's scope of ambition. A similar argument exists for assembly language programming. Thus there is the need for a more powerful and user-oriented programming language.

In considering a higher level language for the general purpose operating system for the ASQ/114, an analysis of the following factors was required: the intended users, size of the language compiler, ease of use of the language, and the anticipated user program types. Briefly the results of that analysis are: the general purpose user of the system will generally have little if any programming experience. These users will primarily be the administrative managers within the aviation community. Their main use of the system is anticipated to be largely confined to the handling of the administrative records and files as well as in the maintenance of various supply records. As such there will probably be a mixture of computer tasks ranging from pure bookkeeping and the arithmetic functions to file handling and file maintenance. With the limited computer memory size available (65K words), the size of the language compiler and its resources must be compatible with the CP 901 hardware. Thus the general purpose system's primary user language

must be reasonably short, simple to use, and capable of handling both arithmetic and file manipulation tasks.

In order to meet these requirements, it was decided that the basic system language should be FORTRAN. FORTRAN has the arithmetic capability required and can as well manipulate files in an efficient manner. Also it is the most universally known computer language and is relatively simple to use.

Once FORTRAN was chosen, it was then necessary to restrict the scope of the language to be implemented in order to provide all the essential features while remaining within the size limitations of the ASQ/114 hardware. Implementation of the entire set of functions available in IBM FORTRAN IV is impractical. It is impractical in that at least 30% of the functions either are not compatible with the system hardware or are normally not utilized by the general purpose user. Such functions as NAMELIST, BLOCK DATA, EXTERNAL, EQUIVALENCE, and FIND are seldom used. Implied do loops, multiple subscripted arrays, and double precision arithmetic can significantly expand the language compiler's size. The sophisticated data formats associated with complex arithmetic operations are not required for the intended users of the proposed system. With these considerations and others in mind the BNF syntax of a FORTRAN compiler has been designed.

The major features of this version of FORTRAN are:

- A. Integer arithmetic
- B. Read and Write statements with variable format
- C. IF Statement with boolean and comparison expressions
- D. DO statement with variable arguments
- E. GO TO statements
- F. DATA statements
- G. Procedure and Function calls
- H. Single subscripted arrays

This version of FORTRAN has many limitations, some of specific design origin others from efforts to insure ease of implementation. There are as well some relaxations from the standard FORTRAN language. The specific characteristics of this FORTRAN language have been placed into four categories for analysis: severe limitations, restrictions, minor inconveniences, and standard relaxations.

A. SEVERE LIMITATIONS

1. integer arithmetic only

B. RESTRICTIONS

1. single subscripted arrays
2. no COMMON, GLOBAL, EQUIVALENCE etc. statements
3. few built in functions
4. modified data statement format

C. MINOR INCONVENIENCES

1. all labels are numbers and are followed by a colon
2. COMMENT is a reserve word
3. termination read (END -) statement eliminated
4. only single precision arithmetic
5. computed/arithmetic GO TO eliminated
6. no continuation statements

D. RELAXATIONS

1. program input is free field (no column dependence)
2. variable names are alphanumeric

The effect of the severe limitation can be minimized by the efficient programmer who is aware of this deficiency. For the anticipated applications of the general purpose system within the aviation community, the restrictions are of little concern. The inconveniences and relaxations should produce nothing more than possible syntax errors in programming and thus do not affect the programmer's logic patterns in his programs. Thus the compiler produced from this syntax appears to well provide the general purpose user of the ASQ/114 system the necessary and desirable capabilities to effectively program and utilize the system.

The compiler's BNF syntax was designed for use in a table driven syntax directed compiler. The generation steps for this compiler resemble

the generation of the CP 901 assembler. The main difference between the two generations is that the total effort required to produce the initial simple compiler is significantly greater than that expended for the generation of the assembler.

The first step for the compiler generation is the development of a syntax analyzer. This analyzer performs a dual task. It firstly analyzes the BNF syntax to insure that the syntax is logically correct and no ambiguities exist. It then generates the parsing tables. The parsing tables are merely the decision tables of the compiler's grammar checker. These tables direct the checker to either hold the current program input symbol or to combine it with previous symbols. After the compiler scans the entire program input, producing an intermediate form of the program, then the second stage of the compiler is completed. All that remains to be done is to develop a set of code generators that emit the proper machine code for each input statement encountered during the scanning.

Thus there are three basic routines that are essential for the generation of a compiler. There must be a syntax analyzer that accepts a BNF syntax input and generates a set of parsing decision tables. Secondly there must be a syntax checker of the program input to verify that the input is syntactically in agreement with that of the BNF grammar. Finally for each BNF reduction in the checking routine, there must be a generator of the proper machine code for that reduction.

For the ASQ/114 system there is only one language to write the three compiler routines, the developed assembly language. It was for this initial generation of the FORTRAN compiler that the assembler was developed. Once these routines are written and the initial compiler is generated, then by the use of the bootstrapping technique newer versions of the

compiler can be developed. Eventually after many incremental additions and subsequent executions, the full range of the compiler as described by the syntax will be implemented.

IX. THE CONTROL OPERATING SYSTEM

A system control program is a set of computer routines that relieves the computer user of organizing and allocating all of the system resources (memory, registers, tapes, and I/O devices) for the tasks he wishes to execute. The control program automatically schedules and supervises the active task, controls the location and retrieval of data, and assures the most efficient processing of tasks through the system. A control program usually consists of several routines of which the Supervisor (monitor), Scheduler, and the I/O Controller are the major ones.

The Supervisor usually controls the actions of the other two, as well as other service routines, language processors and user programs. The service routines consist of housekeeping or utility routines and special application programs. Language processors may be either assemblers, interpreters, translators or compilers. At the present time the CP 901 general purpose system will not have any special application or utility routines nor any special housekeeping programs, other than those required by the Scheduler in manipulating the various tasks within the system. There will be an assembler, used primarily for system development, and a simple FORTRAN compiler.

The basic function of the Scheduler is the efficient manipulation of the system as directed by the general task description supplied by the user. Thus for simple task statements the Scheduler under the control of the Supervisor, will efficiently co-ordinate the necessary resources for the task execution. The Scheduler further interacts with

the I/O Controller to allocate the necessary I/O devices required by the task. The operation of these routines is totally unseen by the user.

All of these routines will constitute a simple Operating System. For the CP 901 the operating system will generally be small as compared to some of the operating systems of the large systems in use today, but none-the-less it should be as effective. This operating system will exploit the resources of the system enabling the user to solve a wide variety of tasks efficiently with the central processor.

Though difficult to estimate, it is believed that only about five hundred instructions will be required for the CP 901 operating system. These routines will permanently reside in memory beginning at address 620 octal, the current loading point of the bootstrap loader of the central processor.

The operating system will have access to one of the tape drives on which the assembler and compiler will reside. If the assigned task requires either of these routines, they must be read into a reserved section of memory and executed utilizing the designated input source and device. This source could either be one of the magnetic tapes or the system's keyboard terminal. If the source is on tape, then the operating system must allocate the second tape drive for input to the task. At this time then the compiler is read into core memory and its tape removed so that this tape drive may be utilized for the object code output from the compiler. Note that if the compiler source is from the keyboard terminal, the second tape drive may be used for the compiler output. Once the output from the compiler is collected on tape, the monitor (Supervisor) regains control of the central processor and awaits a new task. Perhaps the new task may be the execution of previously

compiled program. In this case the compiled program would be read from tape and the necessary I/O devices would be allocated as specified in the task statement. Execution of the program would then begin. Upon completion, the monitor would once again resume control of the processor.

In this manner all processing would be under the control of the operating system as directed by the task assignments of the user. These task assignments will be made using a standard and simple user oriented command language.

This operating system will relieve the user from many of the functions normally required of a general system user on a machine of this size and, in this manner, the CP 901 may be utilized to a much greater extent by the potential squadron users.

X. CONCLUSIONS AND RECOMMENDATIONS

Sufficient computer power is readily available in the CP 901 idle time to warrant efforts to utilize that time. There is also a rapidly expanding administrative and supply workload within the squadrons having this idle computer power to warrant these efforts. Providing the general purpose capability for the CP 901 that is installed in the P3C aircraft both allows utilization of the computer power and assists in the handling of the administrative workload.

Although the feasibility study was briefly conducted, it provided the only hardware system limitation, the lack of sufficient I/O equipment. The study's conclusion indicated that a general purpose operating system was feasible even with the I/O limitation.

The translator operates under the OS/360 operating system and requires 350K for execution. This routine has been tested only with controlled test packages. No known discrepancies exist in this design tool.

The CP 901 assembler presently executes under OS/360 or CP/CMS and produces CP 901 machine code for the various assembly language inputs. This routine has been tested using only controlled test packages. It is not totally coded but those uncoded assembly instruction were designated for future use in section VI. It is believed that the coded set of instructions represent an initial usable set in which reasonable programs could be coded. ✓

The bootstrapping technique described in section VII and appendix B is brief but technically sound. Further description of this process is available in section 8.13 of Reference 3.

The design of the FORTRAN compiler appears reasonable and seems to adequately provide the required system capabilities. It is expected that significant efforts will still be required to produce a working CP 901 FORTRAN compiler.

Once the compiler, assembler, and the control operating system are operational, these software programs may then be combined to produce the end product, the ASQ/114 general purpose digital computer system.

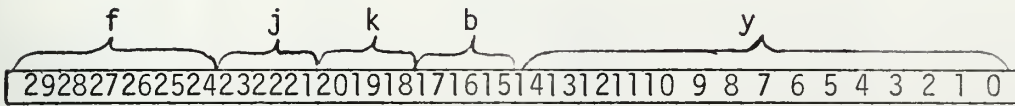
It is recognized that only the basic essentials have been investigated and reported. Many gaps exist and can only be filled with additional dedicated study of the system. It is further essential that this study be undertaken prior to any effort towards the installation of the general purpose ASQ/114 operating system.

It is recommended that followup efforts towards the implementation of the general purpose operating system be initiated either as a followup thesis study or a sponsored project. In this way implementation of a significantly beneficial package may be made available to all P3C squadrons.

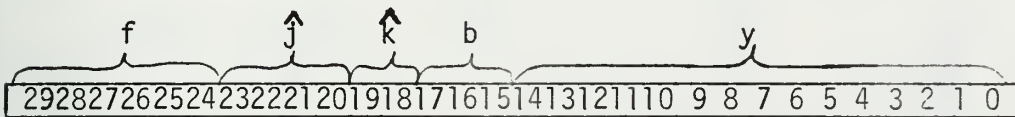
APPENDIX A

Instruction Word Format Types

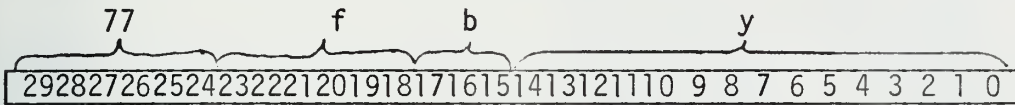
FORMAT I GENERAL INSTRUCTION



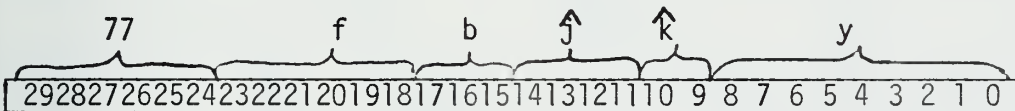
FORMAT I I/O INSTRUCTION WORD



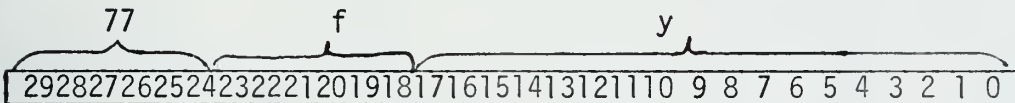
FORMAT II GENERAL INSTRUCTION WORD



FORMAT II SPECIAL INSTRUCTION WORD



FORMAT II DIRECT ADDRESSING INSTRUCTION WORD



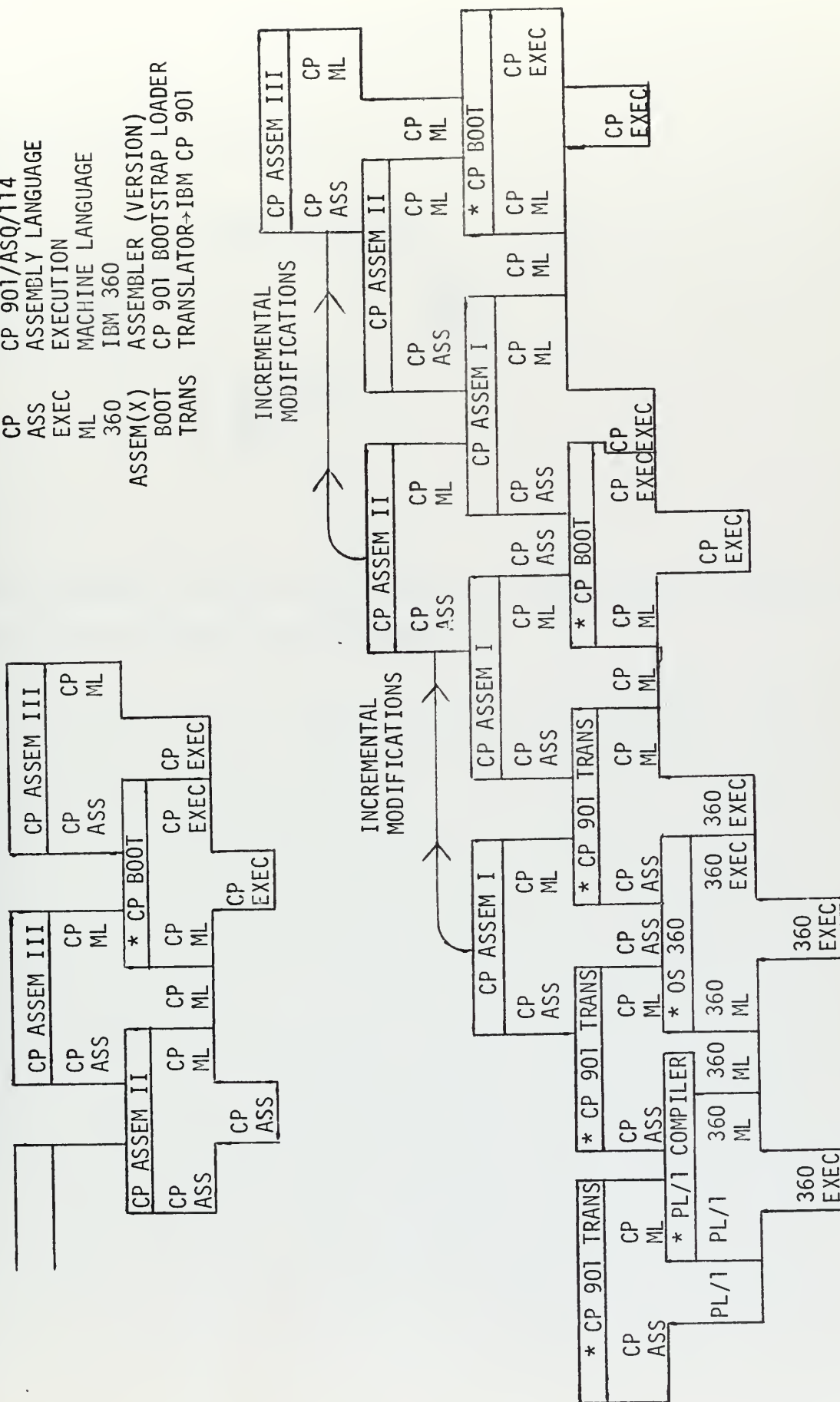
f --- opcode
y --- operand address
b --- b register designator

j --- j designator
k --- k designator
j (with arrow) --- j designator
k (with arrow) --- k designator

Bootstrapping Generation Scheme

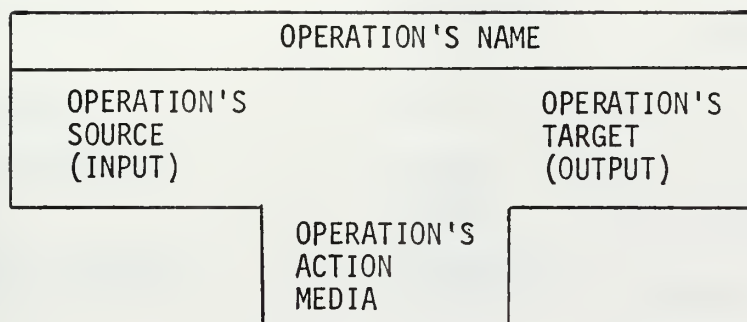
LEGEND
OPERATIONAL UNIT
 CP 901/ASQ/114
 ASSEMBLY LANGUAGE
 EXECUTION
 MACHINE LANGUAGE
 IBM 360
 ASSEMBLER (VERSION)
 CP 901 BOOTSTRAP LOADER
 TRANSLATOR→IBM CP 901

 CP
 ASS
 EXEC
 ML
 360
 ASSEM(X)
 BOOT
 TRANS



APPENDIX B (Continued)

Operation's Structure Description



Note that wherever the symbol (ML) appears, that task step could then be applied to the translator for test execution.

APPENDIX C

Assembler BNF Syntax

[ASSEMBLY]	::=	[ASSEMBLY SET] END
[ASSEMBLY SET]	::=	[STATEMENT] [ASSEMBLY SET] [STATEMENT]
[STATEMENT]	::=	[BASIC STATEMENT]
[BASIC STATEMENT]	::=	[NORMAL STATEMENT] [LABEL] : [NORMAL STATEMENT]
[NORMAL STATEMENT]	::=	[OPERATION] [OPERATION] [OPERAND]
[OPERATION]	::=	ADD SUB MUL DIV EXP LES EQL LEQ NEQ GEQ GTR NEG NOT AND BOR ONE TWO LOD STO DEL STD XIT BFN BFC BBC BRS BSC NOP PRO RTN GET RET RDV WRV WRS DMP TAB ROW SCR SAV UNS SUP LIT EOF DEL DUP XCH 048
[LABEL]	::=	540 541 542 543 544 545 546 547 550 551 552 553 554 555 556 557 560 561 562 563 564 565 566 567 570 571 572 573 574 575 576 577
[OPERAND]	::=	[NUMBER]
[NUMBER]	::=	[DIGIT] [NUMBER] [DIGIT]
[DIGIT]	::=	0 1 2 3 4 5 6 7 8 9

APPENDIX D

FORTRAN Compiler BNF Syntax

[PROGRAM]	::= [STRUCTURE]
[STRUCTURE]	::= [STATEMENT LIST] END
[STATEMENT LIST]	::= [STATEMENT] [STATEMENT LIST] [STATEMENT]
[STATEMENT]	::= [SIMPLE STATEMENT] COMMENT [ANYTHING] [LABEL] : [SIMPLE STATEMENT] [LABEL] : CONTINUE [PROCEDURE DECLARATION]
[ANYTHING]	::= [BETA SET]
[LABEL]	::= [NUMBER]
[ALPHA SET]	::= [CHARACTER SET] [ALPHA SET] [CHARACTER SET]
[BETA SET]	::= [CHARACTER] [BETA SET] [CHARACTER]
[CHARACTER SET]	::= [DIGIT] [LETTER]
[CHARACTER]	::= [DIGIT] [LETTER] [SYMBOL]
[NUMBER]	::= [DIGIT] [NUMBER] [DIGIT]
[DIGIT]	::= 0 1 2 3 4 5 6 7 8 9
[LETTER]	::= A B C D E F G H I etc.
[SYMBOL]	::= + = * ~) (' _ \$ # " / ? @ . , ; :
[SIMPLE STATEMENT]	::= [DIMENSION STATEMENT] [READ STATEMENT] [WRITE STATEMENT] [ASSIGNMENT STATEMENT] [IF STATEMENT] [DATA STATEMENT] [FORMAT STATEMENT] [DO STATEMENT] [CONTROL STATEMENT]

[ASSIGNMENT STATEMENT	::=	[VARIABLE] = [RIGHT PART]
[RIGHT PART]	::=	[EXPRESSION] [SUBSCRIPTED VARIABLE]
[EXPRESSION]	::=	[ARITHMETIC EXPRESSION] [VARIABLE] [FUNCTION CALL]
[ARITHMETIC EXPRESSION]	::=	[TERM] [ARITHMETIC EXPRESSION] + [TERM] [ARITHMETIC EXPRESSION] - [TERM] ~ [TERM]
[TERM]	::=	[PRIMARY] [TERM] * [PRIMARY] [TERM] / [PRIMARY] [TERM] [INVOLUTION FACTOR]
[INVOLUTION FACTOR]	::=	** [PRIMARY]
[PRIMARY]	::=	[VARIABLE] [NUMBER] ([EXPRESSION])
[VARIABLE]	::=	[IDENTIFIER] [SUBSCRIPTED VARIABLE]
[IDENTIFIER]	::=	[ALPHA SET]
[SUBSCRIPTED VARIABLE]	::=	[SUBSCRIPTED HEAD] [EXPRESSION])
[SUBSCRIPT HEAD]	::=	[IDENTIFIER] (
[DIMENSION STATEMENT]	::=	DIMENSION [ARRAYS]
[ARRAY]	::=	[SUBSCRIPT HEAD] [ARRAY LIST]
[ARRAYS]	::=	[ARRAY] [ARRAYS] , [ARRAY]
[ARRAY LIST]	::=	[NUMBER])
[READ STATEMENT]	::=	[READ HEAD]) [VARIABLE LIST]
[READ HEAD]	::=	READ ([IO]
[IO]	::=	[NUMBER] , [LABEL]
[VARIABLE LIST]	::=	[VARIABLE] [VARIABLE LIST] , [VARIABLE]

[WRITE STATEMENT]	::=	[WRITE HEAD]) [EXPRESSION LIST] [WRITE HEAD]) [STRING LIST]
[WRITE HEAD]	::=	WRITE ([IO]
[EXPRESSION LIST]	::=	[EXPRESSION] [EXPRESSION LIST] , [EXPRESSION]
[STRING LIST]	::=	[STRING] [STRING LIST] , [STRING]
[STRING]	::=	' [STRING BODY] '
[STRING BODY]	::=	[BETA STRING]
[IF STATEMENT]	::=	[BOOLEAN PRIMARY] [BOOLEAN EXPRESSION] .AND. [BOOLEAN PRIMARY] [BOOLEAN EXPRESSION] .OR. [BOOLEAN PRIMARY] .NOT. [BOOLEAN PRIMARY]
[BOOLEAN PRIMARY]	::=	[COMPARISON] ([BOOLEAN EXPRESSION])
[COMPARISON]	::=	[EXPRESSION] [RELATION] [EXPRESSION]
[RELATION]	::=	.EQL. .NEQ. .LES. .LEQ. .GTR. .GEQ.
[THEN]	::=	[JUMP] [ASSIGNMENT STATEMENT] [PROCEDURE CALL] STOP
[JUMP]	::=	[GO TO STATEMENT]
[DATA STATEMENT]	::=	DATA [DATA MODE]
[DATA MODE]	::=	[DATA DEF] [DATA MODE] , [DATA DEF]
[DATA DEF]	::=	[ALPHA SET] / [INFO]
[INFO]	::=	[NUMBER]
[FORMAT STATEMENT]	::=	[LABEL] : [FORMAT HEAD]
[FORMAT HEAD]	::=	FORMAT ([FORMAT LIST]
[FORMAT LIST]	::=	[FORMAT FIELD])
[FORMAT FIELD]	::=	[FORMAT EXPRESSION] [FORMAT FIELD] , [FORMAT EXPRESSION]

[FORMAT EXPRESSION]	::=	[ALPHA SET] [MULTIPLE] [ALPHA SET]
[MULTIPLE]	::=	[NUMBER]
[GO TO STATEMENT]	::=	GO TO [LABEL] GOTO [LABEL]
[CONTROL STATEMENT]	::=	STOP RETURN
[DO STATEMENT]	::=	DO [DO EXPRESSION]
[DO EXPRESSION]	::=	[LABEL] [DO FORMAT]
[DO FORMAT]	::=	[VARIABLE] = [DO LIST]
[DO LIST]	::=	[ARG] , [ARG LIST]
[ARG LIST]	::=	[ARG] [ARG] , [ARG]
[ARG]	::=	[VARIABLE]
[PROCEDURE DECLARATION]	::=	[PROCEDURE HEAD] [PROCEDURE BODY]
[PROCEDURE HEAD]	::=	FUNCTION [PROCEDURE LIST] PROCEDURE [PROCEDURE LIST]
[PROCEDURE LIST]	::=	[PROCEDURE NAME] ([ARGUMENTS]
[PROCEDURE NAME]	::=	[ALPHA SET]
[ARGUMENTS]	::=	[ARGS])
[ARGS]	::=	[ARG] [ARGS] , [ARG]
[PROCEDURE BODY]	::=	[STRUCTURE]
[FUNCTION CALL]	::=	[PROCEDURE NAME] ([ARGUMENTS]
[PROCEUDRE CALL]	::=	CALL [PROCEDURE LIST]

CP 901 TRANSLATOR LISTING

```

//RAY$SIM JOB (0369,0405NT,CS04),'DENNIS E. RAY',TIME=4
//EXEC PL1CLG,REGION.GO=350K
//PL1L:SYSIN DD *
SIMUL: PROCEDURE OPTIONS (MAIN);
DCL SYSIN FILE STREAM PRINT ENV (F(133));
DCL (A,A1,Q,Q1,U,Z,Y_BAR,INPUT) BIT (30);
DCL (F,FS) BIT (6);
DCL AQ BIT (60);
DCL (J,K,B,R) BIT (3);
DCL (L,P_DEC,P_OCT,F_DEC,A_VAL,Q_VAL,YBAR_VAL) FIXED BINARY (15);
DCL (B_DEC,J_DEC,K_DEC,R_DEC) FIXED BINARY (15);
DCL J_HAT BIT (4);
DCL K_HAT BIT (2);
DCL (P,Y) BIT (15);
DCL Y_1 BIT (9);
DCL Y_2 BIT (16);
DCL INPUT4 CHAR (30) VAR;
DCL INPUT1 CHAR (30);
DCL PARITY CHAR (4);
DCL BSTR BIT (1) INITIAL ('0'B);
DCL MEM1 (0:32767) BIT (30);
DCL MEM2 (0:32767) BIT (30);
DCL FLAG BIT (1) INITIAL ('0'B);
DCL (ASTR,CSTR) BIT (30) VAR;
DCL VAR (77) LABEL;
DCL J_VAR (7) LABEL;
DCL B_REGIS (0:7) BIT (15);
DCL (JUMP1,JUMP2,JUMP3,STOP5,STOP6,STOP7) BIT (1) INITIAL ('0'B);

/* PROCEDURE OCTAL RETURNS THE FIXED DEC REPRESENTATION
   OF THE THREE BIT INPUT IN OCTAL NOTATION. */

/* PROCEDURE REP_STR REPEATS THE FIRST PARAMETER SINGLE
   BIT THE NUMBER_OF TIMES SPECIFIED BY THE SECOND PARAMETER. */

/* PROCEDURE MEMORY IS USED TO ADDRESS THE UPPER AND
   LOWER HALF OF MEMORY BY THE SINGLE REFERENCE TO
   DUMMY MEMORY. */

/* PROCEDURE VAL_THIRTY RETURNS A FIXED BINARY NUMBER
   REPRESENTED IN DECIMAL FORM IN OCTAL NOTATION. */

/* PROCEDURE VAL_15 RETURNS A FIXED BINARY NUMBER IN OCTAL
   NOTATION FOR THE FIFTEEN BIT INPUT. */

/* PROCEDURE PAR_CHECK IS A PARITY CHECKER IN THAT IT
   COUNTS THE NUMBER OF 1 BITS IN A GIVEN BIT STRING

```

SIM00010
SIM00020

SIM00050
SIM00060
SIM00070
SIM00080
SIM00090
SIM00100
SIM00110
SIM00120
SIM00130
SIM00140
SIM00150
SIM00160

SIM00170
SIM00180
SIM00190
TIM01320


```

OF THIRTY BITS AND RETURNS EITHER EVEN OR ODD.          */

/* PROCEDURE CIR_SFT CIRCULARLY SHIFTS THE THIRTY BIT
   INPUT STRING THE NUMBER OF BITS SPECIFIED BY THE
   SECOND PARAMETER.                                     */

/* PROCEDURE REV_OCTAL RETURNS THE THREE BITS THAT
   REPRESENT THE INPUT NUMERIC CHARACTER                */

/* PROCEDURE STORE_STORES INTO THE DUMMY MEMORY THE FIRST
   PARAMETER AT THE ADDRESS GIVEN BY THE SECOND
   PARAMETER.                                           */

/* PROCEDURES J_CHANGE READ IT SPEC_J REP_STO REP_READ
   ALL MANIPULATE THE VARIOUS BIT PATTERNS AS DIRECTED
   BY THE MODIFIERS. EACH OF THESE PROCEDURES ACCOMPLISHES
   THE ACTUAL MODIFICATION OF ONE OF THE MODIFIERS.    */

DCL OCTAL ENTRY (BIT (3)) RETURNS (FIXED BINARY);
DCL REP_STR ENTRY (BIT(1), FIXED BINARY (15)) ;
DCL MEMORY ENTRY (FIXED BINARY (15)) RETURNS (BIT (30));
DCL VAL_THIRTY ENTRY (BIT (30)) RETURNS (FIXED BINARY (15));
DCL VAL_15 ENTRY (BIT(15)) RETURNS (FIXED BINARY (15));
DCL PAR_CHECK ENTRY (BIT (30)) RETURNS (CHAR (4));
DCL CIR_SFT ENTRY (BIT (30), FIXED BINARY (15)) RETURNS (BIT (30));
DCL REV_OCTAL ENTRY (CHAR (1)) RETURNS (BIT (3));
DCL STORE ENTRY (BIT (30), BIT (15));
DCL J_CHANGE ENTRY;
DCL READ_IT ENTRY;
DCL SPEC_J ENTRY (BIT (30), BIT (15));
DCL REP_STO ENTRY;
DCL REP_READ ENTRY;
DCL (HOLD1, HOLD2, HOLD3, HOLD4, HOLD5) FIXED BINARY (15);
DCL TEMP1 CHAR (30) VAR;
DCL TEMP2 CHAR (30) VAR;

START:

/* INITIALIZE B REGISTERS TO 0 */

DO L = 0 TO 7;
  B_REGIS (L) = '0000000000000000'B;
END;

/* P HOLDS THE ADDRESS OF THE NEXT INSTRUCTION TO BE
   PROCESSED. P_DEC IS USED TO INCREMENT THE ADDRESS
   P AND IS INTERPRETED BY THE PROCEDURE (VAL_15)

```

```

SIM000210
SIM000220
SIM000230
SIM000240
SIM000250
SIM000260
SIM000270
SIM000280
SIM000290
SIM000300
SIM000310
TIM001900
SIM000320
SIM000330
SIM000340
SIM000350
SIM000360
SIM000370
TIM01230

```

```

TIM01350
TIM01360
TIM01370
TIM01250
TIM01400
TIM01410
TIM01420

```



```

FOR CONVERSION TO OCTAL REPRESENTATION. U HOLDS THE
CONTENTS OF THE ADDRESS SPECIFIED BY P AS THE
INSTRUCTION IS EXECUTED ACCORDING TO THE VARIOUS
FIELDS CONTAINED IN THE INSTRUCTION.
P = '000000110010000'B;
P_OCT = 620;
P_DEC = 400;
START2: PUT FILE (SYSPRINT) EDIT ('P = ',P) (SKIP,A,B(15));
PUT FILE (SYSPRINT) EDIT ('P DEC IS ',P_DEC) (SKIP,A,F(6));
PUT FILE (SYSPRINT) EDIT ('P OCT = ',P_OCT) (SKIP,A,F(6));
START3: U = MEMORY (P_OCT);
PUT FILE (SYSPRINT) EDIT ('U = ',U) (SKIP,A,B(30));
F = SUBSTR (U,1,6);
B = SUBSTR (U,13,3);
IF F = '111111' THEN GO TO DIRECT;
J_HAT = SUBSTR (U,7,3);
K_HAT = SUBSTR (U,10,3);
Y = SUBSTR (U,16,15);
GO TO PROCESS;
DIRECT: FS = SUBSTR (U,7,6);
J_HAT = SUBSTR (U,16,4);
K_HAT = SUBSTR (U,20,2);
Y_1 = SUBSTR (U,22,9);
Y_2 = SUBSTR (U,14,16);
PROCESS: F_DEC = 10 * OCTAL (SUBSTR (F,1,3)) + OCTAL (SUBSTR (F,4,3));
PUT FILE (SYSPRINT) EDIT ('F DEC = ',F_DEC) (SKIP,A(8),F(8));
PUT FILE (SYSPRINT) EDIT ('A REG IS ',A) (SKIP,A,B(30));
PUT FILE (SYSPRINT) EDIT ('Q REG IS ',Q) (SKIP,A,B(30));
IF F_DEC = 0 THEN DO;
P_DEC = P_DEC + 1;
P = BIT (P_DEC,15);
P_DEC = VAL_15(P);
GO TO START2;
END;
/* THIS CREATES THE OPERAND (Y) FROM Y AND B OF THE
INSTRUCTION WORD.
Y = BIT (BINARY (Y,15,0) + BINARY (B_REGIS (OCTAL (B)),15,0),15);
GO TO VAR (F_DEC);
*/
/* RIGHT SHIFT Q REGISTER
HOLD1 IS THE SHIFT COUNTER
ASTR CONTAINS THE PROPER NUMBER OF ZERO BITS
VAR (1): CALL READ_IT;

```

TIM01430
TIM01440
TIM01450
TIM01460
*/

SIM00580
SIM00600
SIM00620

SIM00640
SIM00710
SIM00650
SIM00670
SIM00680
SIM00690
SIM00700
SIM00720
SIM00730

SIM00760
SIM00770
SIM00780
SIM00790
SIM00800

SIM00820
SIM00840
SIM00850
SIM00860
SIM00870
SIM00880
TIM01320
TIM01330

SIM00890
SIM00900
TIM00010
TIM00020
TIM00030
TIM00410
SIM00910


```

HOLD1 = 10 * OCTAL (SUBSTR (Y_BAR,25,3)) +
OCTAL (SUBSTR (Y_BAR,28,3));
CALL J_CHANGE;
CALL REP_STR (BSTR,HOLD1);
Q = ASTR || SUBSTR (Q,1,30-HOLD1);
GO TO ENDING;

/*
RIGHT SHIFT A REGISTER
HOLD1 IS THE SHIFT COUNTER
ASTR CONTAINS THE PROPER NUMBER OF ZERO BITS */
VAR (2): CALL READ_IT;
CALL J_CHANGE;
HOLD1 = 10 * OCTAL (SUBSTR (Y_BAR,25,3)) +
OCTAL (SUBSTR (Y_BAR,28,3));
CALL REP_STR (BSTR,HOLD1);
A = ASTR || SUBSTR (A,1,30-HOLD1);
GO TO ENDING;

/*
RIGHT SHIFT A & Q REGISTERS AS ONE REGISTER
HOLD1 IS THE SHIFT COUNTER
TEMP1 HOLDS THE BITS TO BE SHIFTED INTO THE Q REGISTER
ASTR CONTAINS THE PROPER NUMBER OF ZERO BITS */
VAR (3): CALL READ_IT;
CALL J_CHANGE;
HOLD1 = 10 * OCTAL (SUBSTR (Y_BAR,25,3)) +
OCTAL (SUBSTR (Y_BAR,28,3));
CALL REP_STR (BSTR,HOLD1);
TEMP1 = SUBSTR (A,30-HOLD1,HOLD1);
A = ASTR || SUBSTR (A,1,30-HOLD1);
Q = TEMP1 || SUBSTR (Q,1,30-HOLD1);
GO TO ENDING;

/*
THIS INSTRUCTION COMPARES THE VALUE OF Y_BAR WITH THE
VALUE OF THE A OR Q REGISTER. THE BRANCH_CONDITION DESIGNATOR
J_DEC DETERMINES WHETHER THE NEXT INSTRUCTION IS TO BE
SKIPPED OR NOT IN COMPARISON TO THE VARIOUS VALUES. */
VAR (4): CALL READ_IT;
J_DEC = OCTAL (J);
YBAR_VAL = VAL_THIRTY (Y_BAR);
A_VAL = VAL_THIRTY (A);
Q_VAL = VAL_THIRTY (Q);
IF J_DEC = 0 THEN GO TO ENDING;
IF J_DEC = 1 THEN GO TO SKIP;
GO TO J_VAR (J_DEC);
J_VAR (2): IF YBAR_VAL -> Q_VAL THEN GO TO SKIP;

```


SIM01410
SIM01420
SIM01430
SIM01440
SIM01450
SIM01460
SIM01470
SIM01480
SIM01490
SIM01500
SIM01510
SIM01520
SIM01530
SIM01540
SIM01550
SIM01560
SIM01570
TIM00470
TIM00480
TIM00490
TIM00300
SIM01580
SIM01590
TIM00430
TIM00440
TIM00450
SIM01700
SIM01710
TIM00580
TIM00590
TIM00600
TIM00570
SIM01720
SIM01730
TIM00540
TIM00550
TIM00560
SIM01810
SIM01820
TIM00720
TIM00730
TIM00740
TIM00750
TIM00700
SIM01830
SIM01840
TIM00640
TIM00650

```

GO TO ENDING;
J_VAR (3): IF YBAR_VAL > Q_VAL THEN GO TO SKIP;
GO TO ENDING;
J_VAR (4): IF (Q_VAL < YBAR_VAL & YBAR_VAL > A_VAL)
            THEN GO TO SKIP;
GO TO ENDING;
J_VAR (5): IF (YBAR_VAL > Q_VAL | YBAR_VAL -> A_VAL)
            THEN GO TO SKIP;
GO TO ENDING;
J_VAR (6): IF YBAR_VAL -> A_VAL THEN GO TO SKIP;
GO TO ENDING;
J_VAR (7): IF YBAR_VAL > A_VAL THEN GO TO SKIP;
GO TO ENDING;
SKIP: P_DEC = P_DEC + 1;
P = BIT (P_DEC, 15);
GO TO ENDING;

```

/* THIS INSTRUCTION SHIFTS REGISTER Q CIRCULARLY
TO THE LEFT (HOLD1) POSITIONS. THE
PROCEDURE (CIR_SFT) ACCOMPLISHES THE SHIFT.

*/

```

VAR (5): CALL READ_IT;
CALL J_CHANGE;
HOLD1 = 10 * OCTAL (SUBSTR (Y_BAR, 25, 3)) +
          OCTAL (SUBSTR (Y_BAR, 28, 3));
Q = CIR_SFT (Q, HOLD1);
GO TO ENDING;

```

/* THIS INSTRUCTION SHIFTS REGISTER A CIRCULARLY
TO THE LEFT (HOLD1) POSITIONS. THE PROCEDURE
(CIR_SFT) ACCOMPLISHES THE SHIFT.

*/

```

VAR (6): CALL READ_IT;
CALL J_CHANGE;
HOLD1 = 10 * OCTAL (SUBSTR (Y_BAR, 25, 3)) +
          OCTAL (SUBSTR (Y_BAR, 28, 3));
A = CIR_SFT (A, HOLD1);
GO TO ENDING;

```

/* THIS INSTRUCTION SHIFTS THE A & Q REGISTERS TO THE
CIRCULARLY AS ONE SIXTY BIT REGISTER. THE UPPER
BITS OF A (TEMP1) ARE SHIFTED INTO THE LOWER BITS
BITS OF Q.

*/

```

VAR (7): CALL READ_IT;
CALL J_CHANGE;
HOLD1 = 10 * OCTAL (SUBSTR (Y_BAR, 25, 3)) +
          OCTAL (SUBSTR (Y_BAR, 28, 3));

```



```

TEMP1 = SUBSTR (A,1,HOLD1);
A = SUBSTR (A,HOLD1 + 1); SUBSTR (Q,1,HOLD1);
Q = SUBSTR (Q,HOLD1 + 1); TEMP1;
GO TO ENDING;

/* THIS INSTRUCTION ENTERS THE VALUE Y_BAR INTO THE
   Q REGISTER. */

VAR (10): CALL READ_IT;
          CALL J_CHANGE;
          Q = Y_BAR;
          GO TO ENDING;

/* THIS INSTRUCTION ENTERS THE VALUE Y_BAR INTO THE
   A REGISTER. */

VAR (11): CALL READ_IT;
          CALL J_CHANGE;
          A = Y_BAR;
          GO TO ENDING;

/* THIS INSTRUCTION TRANSMITS THE VALUE OF THE LOWER
   FIFTEEN BITS OF THE OPERAND ADDRESS (Y_BAR) INTO
   THE B REGISTER DESIGNATED BY THE VALUE OF J (J_DEC). */

VAR (12): CALL READ_IT;
          J_DEC = OCTAL (J);
          B_REGIS (J_DEC) = SUBSTR (Y_BAR,16,15);
          GO TO ENDING;

/* THIS INSTRUCTION ESTABLISHES ONE WORD OF OUTPUT
   (MEMORY (Y)) AND DESIGNATES THE OUTPUT CHANNEL (J_HAT).
   THE ADDRESS Y IS STORED IN LOCATION 140 PLUS J_HAT.
   FOR THIS IMPLEMENTATION ONE WORD IS PLACED INTO FILE
   SYSPRINT WITH APPROPRIATE IDENTIFICATION. MONITOR
   AND FORCE FUNCTIONS AS DESIGNATED BY K HAT ARE
   IGNORED FOR IMPLEMENTATION. */

VAR (13): PUT FILE (SYSPRINT) EDIT ('OUTPUT BUFFER CREATION') (SKIP,A);
          MEM1 (140 + (OCTAL ('00'B) SUBSTR (J_HAT,1,1)) * 10 +
          OCTAL (SUBSTR (J_HAT,2,3))) = MEMORY (VAL_15 (Y));
          PUT FILE (SYSPRINT) EDIT
          ('OUTPUT MEMORY OF Y', MEMORY (VAL_15(Y)), ' EQUALS ');
          VAL_THIRTY (MEMORY (VAL_15 (Y))) (SKIP,A,B,A,F(8));
          GO TO ENDING;

/* THIS INSTRUCTION STORES THE CONTENTS OF THE Q
   REGISTER AT THE STORAGE ADDRESS Y AS MODIFIED
   BY THE DESIGNATOR K. FOR K = 0 COMPLEMENT REGISTER

```

TIM000670
 TIM000680
 TIM000690
 SIM01880
 SIM01890
 TIM000780
 TIM000790
 TIM000760
 SIM01900
 SIM01910
 SIM01920
 SIM01930
 SIM01940
 TIM000830
 TIM000840
 TIM000820
 SIM01950
 SIM01960
 SIM01970
 SIM01980
 SIM01990
 TIM000900
 TIM000910
 TIM000920
 SIM02000
 TIM000870
 TIM000880
 SIM02020
 SIM02030

SIM02050
 SIM02060
 TIM01040
 TIM01050
 TIM01060


```

VAR (17):PUT FILE (SYSPRINT) EDIT ('STORE CN OT TEST EFB') (SKIP,A);
K_DEC = OCTAL ('0'B || K_KAT);
IF K_DEC = 0 | K_DEC = 1 THEN DO;
PUT FILE (SYSPRINT) EDIT ('INPUT WITH K HAT = 0 OR 1 NOT USED')
(SKIP,A);
GO TO ENDING;
END;
IF K_DEC = 3 THEN
PUT FILE (SYSPRINT) EDIT ('SIMULATE INTERRUPT WORD TRANSFER ')
(SKIP,A);
ELSE DO;
PUT FILE (SYSPRINT) EDIT ('INPUT USING CHANNEL ',J_HAT)
(SKIP,A,B);
HOLD4 = VAL15(Y);
GET FILE (SYSIN) EDIT (INPUT1) (A(30));
IF HOLD4 > 32767 THEN
MEM2 (HOLD4 - 32767) = BIT (INPUY1,30);
ELSE MEM1 (HOLD4) = BIT (INPUT1,30);
END;
GO TO ENDING;

/* THIS INSTRUCTION ADDS TO THE A REGISTER THE CONTENTS
OF THE OPERAND Y_BAR NOTE THE USE OF THE
EXPLICIT CALLS TO THE BINARY AND BIT CONVERTER OF PL/1. */

VAR (20): CALL READ_IT;
CALL J_CHANGE;
A = BIT (BINARY (Y_BAR,31,0) + BINARY (A,31,0),30);
GO TO ENDING;

/* THIS INSTRUCTION SUBTRACTS FROM THE A REGISTER
THE CONTENTS OF THE OPERAND Y_BAR. NOTE THE USE OF
THE EXPLICIT CALLS TO THE BINARY AND BIT CONVERTERS
OF PL/1. */

VAR (21): CALL READ_IT;
A = BIT (BINARY (A,31,0) - BINARY (Y_BAR,31,0),30);
CALL J_CHANGE;
GO TO ENDING;

/* THIS INSTRUCTION MULTIPLIES THE CONTENTS OF THE Q REGISTER
WITH THE K MODIFIED Y_OPERAND AND LEAVES THE RESULT
IN ONE SIXTY BIT REGISTER AQ. A = UPPER THIRTY BITS
Q = LOWER THIRTY BITS */

VAR (22): CALL READ_IT;
AQ = BIT (VAL_THIRTY (Q) * VAL_THIRTY (Y_BAR),60);

```

SIM02290
SIM02300
TIM01460
TIM01470
TIM01480
TIM01450
SIM02310
SIM02320
SIM02330
SIM02340
SIM02350
TIM01520
TIM01530
TIM01540
TIM01550
TIM01560
SIM02360
SIM02370
SIM02380
SIM02390
SIM02400

SIM02410


```

A = SUBSTR (AQ,1,30);
Q = SUBSTR (AQ,31,30);
CALL J_CHANGE;
GO TO ENDING;

/* THIS INSTRUCTION DIVIDES THE DOUBLE LENGTH REGISTER
AQ BY THE K MODIFIED Y OPERAND. THE QUOTIENT IS LEFT
IN THE Q REGISTER, THE REMAINDER IN THE A REGISTER.
FOR K = 7 TAKE THE SQUARE OF AQ LEAVING THE ROOT IN
Q REGISTER AND REMAINDER IN THE A REGISTER.

VAR (23): CALL READ_IT;
AQ = A || Q;
K DEC = OCTAL (K);
IF K DEC = 7 THEN DO;
PUT FILE (SYSPRINT) EDIT ('TAKING SQR') (SKIP,A);
END;
ELSE DO;
PUT FILE (SYSPRINT) EDIT ('DIVIDING AQ') (SKIP,A);
Q = BIT (BINARY (AQ,31,0) / VAL_THIRTY (Y_BAR),30);
END;
GO TO ENDING;

/* THIS INSTRUCTION ADDS THE VALUE OF Y TO THE A
REGISTER AND THEN STORES THE CONTENTS OF THE A
REGISTER AT THE STORAGE ADDRESS DESIGNATED BY
Y AND THE K DESIGNATOR. THE STORING IS ACCOMPLISHED
BY THE CALL TO THE FUNCTION (REP_STO).

VAR (24): A = BIT (BINARY (Y,31,0) + BINARY (A,31,0),30);
CALL J_CHANGE;
R DEC = OCTAL (R);
IF R DEC > 4 & FLAG = '1'B THEN
Y = BIT (BINARY (Y,15,0) + BINARY (B_REGIS (6),15,0),15);
CALL REP_STO (A,Y);
GO TO ENDING;

/* THIS INSTRUCTION SUBTRACTS THE VALUE OF Y FROM
THE (A) REGISTER AND THEN STORES THE VALUE OF
THE (A) REGISTER AT THE STORAGE ADDRESS DESIGNATED
BY Y AND THE K DESIGNATOR. THE STORING IS
ACCOMPLISHED BY THE CALL TO THE PROCEDURE (REP_STO).

VAR (25): A = BIT (BINARY (A,31,0) - BINARY (Y,31,0),30);
R DEC = OCTAL (R);
IF R DEC > 4 & FLAG = '1'B THEN
Y = BIT (BINARY (Y,15,0) + BINARY (B_REGIS (6),15,0),15);
CALL REP_STO (A,Y);

```

SIM02420
SIM02430
SIM02440

*/

SIM02450

SIM02460
SIM02470
SIM01610
SIM01620
SIM01630
SIM01640
SIM01650
SIM01660
SIM01590
SIM02490

*/

SIM02500
SIM02510
SIM02520
SIM01720
SIM01730
SIM01740
SIM01750
SIM01760
SIM01770
SIM01700

SIM02550

SIM02560
SIM02570
SIM02580
TIM01810
TIM01820
TIM01830
SIM02590
SIM02600
SIM02610
SIM02620
SIM02630

/*
*/

CALL J_CHANGE;
GO TO ENDING;
THIS INSTRUCTION ADDS THE VALUE OF THE OPERAND
Y_BAR TO THE CONTENTS OF THE Q REGISTER.

VAR (26): CALL READ IT;
Q = BIT (BINARY (Q,15,0) + BINARY (Y_BAR,15,0),30);
CALL J_CHANGE;
GO TO ENDING;

*/

/* THIS INSTRUCTION SUBTRACTS THE CONTENTS OF THE K MODIFIED
OPERAND Y FROM THE CONTENTS OF THE Q REGISTER.

VAR (27): CALL READ IT;
Q = BIT (BINARY (Q,15,0) - BINARY (Y_BAR,15,0),30);
CALL SPEC J;
GO TO ENDING;

/* THIS INSTRUCTION ADDS THE VALUE OF THE OPERAND Y
TO THE CONTENTS OF THE Q REGISTER AND PLACES THE
RESULT INTO THE A REGISTER. */

VAR (30): A = BIT (BINARY (Q,15,0) + BINARY (Y,15,0),30);
CALL J_CHANGE;
CALL READ IT;
GO TO ENDING;

/* THIS INSTRUCTION SUBTRACTS THE VALUE OF THE
OPERAND Y FROM THE CONTENTS OF THE Q REGISTER AND
THEN COMPLEMENTS THE RESULT AND PLACES IT INTO THE
A REGISTER. */

VAR (31): A = BIT (BINARY (Q,15,0) - BINARY (Y,15,0),30);
CALL J_CHANGE;
A = ~ (A);
CALL READ IT;
GO TO ENDING;

/* THIS INSTRUCTION ADDS THE CONTENTS OF BOTH THE A
AND THE Q REGISTERS PLACES THE RESULT INTO THE A
REGISTER. IT THEN STORES THE CONTENTS OF THE A
REGISTER AT THE STORAGE ADDRESS OF THE OPERAND Y
AS DESIGNATED BY THE DESIGNATOR K. THE STORING
IS ACCOMPLISHED BY THE CALL TO THE PROCEDURE
(STORE). */

VAR (32): A = BIT (BINARY (A,15,0) + BINARY (Q,15,0),30);

SIM02640
TIM01860
SIM02660
SIM02670
TIM02090
TIM02100
TIM02110
TIM02120
SIM02680
SIM02690
SIM02700
SIM02710
SIM02720
TIM02150
TIM02160
TIM02170
TIM02180
TIM02190
SIM02730
SIM02740
SIM02750
SIM02760
SIM02770
SIM02780
TIM02220
TIM02230
TIM02240
TIM02250
TIM02260
TIM02270
TIM02280
TIM02290
SIM02790

SIM02800
SIM02810
SIM02820
SIM02830
TIM02320
TIM02330
TIM02340
TIM02350
TIM02360
TIM02370
SIM02840
SIM02850
SIM02860
SIM02870
TIM00040
TIM00050
TIM00060
TIM00070
TIM00080
TIM00090
TIM00100
SIM02880
TIM00030

```
CALL J_CHANGE;
CALL STORE (A,Y);
GO TO ENDING;
```

```
/*
THIS INSTRUCTION SUBTRACTS THE CONTENTS OF THE RESULT
Q REGISTER FROM THE A REGISTER AND STORES THE RESULT
INTO THE ADDRESS SPECIFIED BY THE OPERAND Y AND THE
DESIGNATOR K. THE STORING IS ACCOMPLISHED BY THE CALL
TO THE PROCEDURE (STORE).
*/
```

```
VAR (33): A = BIT (BINARY (A,15,0) - BINARY (Q,15,0),30);
CALL J_CHANGE;
CALL STORE (A,Y);
GO TO ENDING;
```

```
/*
THIS INSTRUCTION ADDS THE CONTENTS OF Y TO THE
CONTENTS OF THE Q REGISTER AND PLACES THE RESULT
IN THE A REGISTER. THE A REGISTER CONTENTS ARE THEN
STORED AT STORAGE ADDRESS Y. THE STORING IS
ACCOMPLISHED BY THE CALL TO THE PROCEDURE (REP_STO).
(REP_STO) UTILIZES THE K DESIGNATOR AS A MODIFIER.
*/
```

```
VAR (34): A = BIT (BINARY (Q,15,0) + BINARY (Y,15,0),30);
R_DEC = OCTAL (R);
IF R_DEC > 4 & FLAG = '1'B THEN
Y = BIT (BINARY (Y,15,0) + BINARY (B_REGIS (6),15,0),15);
CALL REP_STO (A,Y);
CALL J_CHANGE;
GO TO ENDING;
```

SIM02910
SIM02920
SIM02930
SIM02940
TIM00160
TIM00170
TIM00180
TIM00190
TIM00200
TIM00210
TIM00220
TIM00230
TIM00240
TIM00140
SIM02970

```
/*
THIS INSTRUCTION SUBTRACTS THE CONTENTS OF THE
OPERAND Y FROM THE CONTENTS OF THE Q REGISTER
AND PLACES THE RESULTS INTO THE A REGISTER. THE
CONTENTS OF THE A REGISTER IS STORED AT THE ADDRESS
Y. THE STORING IS ACCOMPLISHED BY THE CALL TO THE
PROCEDURE (REP_STO) WHICH MAKES USE OF THE K
MODIFIER. NOTE THAT REGISTER A IS COMPLETED PRIOR
TO THE CALL TO (REP_STO).
*/
```

```
VAR (35): A = BIT (BINARY (Q,15,0) - BINARY (Y,15,0),30);
R_DEC = OCTAL (R);
IF R_DEC > 4 & FLAG = '1'B THEN
Y = BIT (BINARY (Y,15,0) + BINARY (B_REGIS (6),15,0),15);
CALL REP_STO (A,Y);
CALL J_CHANGE;
GO TO ENDING;
```

SIM02980
SIM02990
SIM03000
SIM03010


```

/*
THIS INSTRUCTION ADDS ONE TO THE CONTENTS OF THE
OPERAND Y AND PLACES THE RESULT INTO THE A REGISTER.
THE CONTENTS OF THE A REGISTER ARE THEN STORED
AT STORAGE ADDRESS Y USING THE CALL TO PROCEDURE
(REP_STO) AS MODIFIED BY THE K DESIGNATOR.
*/
TIM00290
TIM00300
TIM00310
TIM00320
TIM00330
TIM00340
SIM03030

VAR (36): A = BIT (BINARY (Y,15,0) + 1 ,30);
CALL J_CHANGE;
R_DEC = OCTAL (R);
IF R_DEC > 4 & FLAG = '1'B THEN
Y = BIT (BINARY (Y,15,0) + BINARY (B_REGIS (6),15,0),15);
CALL REP_STO (A,Y);
GO TO ENDING;
TIM03040
TIM03050
TIM03060
TIM03090
TIM00400
TIM00410
TIM00420
TIM00430
TIM00440
TIM00370
TIM03080
TIM03090

/*
THIS INSTRUCTION SUBTRACTS THE VALUE OF THE
OPERAND Y FROM THE ONE AND PLACES THE
RESULT INTO THE A REGISTER. THE A REGISTER IS
THEN COMPLEMENTED AND STORED AT THE STORAGE ADDRESS
Y AS MODIFIED BY THE K DESIGNATOR.
*/
TIM03040
TIM03050
TIM03060
TIM03090
TIM00400
TIM00410
TIM00420
TIM00430
TIM00440
TIM00370
TIM03080
TIM03090

VAR (37): A = BIT (1 - BINARY (Y,15,0),30);
A = ~ (A);
CALL J_CHANGE;
R_DEC = OCTAL (R);
IF R_DEC > 4 & FLAG = '1'B THEN
Y = BIT (BINARY (Y,15,0) + BINARY (B_REGIS (6),15,0),15);
CALL REP_STO (A,Y);
GO TO ENDING;
TIM03100
TIM03110
TIM03120
TIM00500
TIM00510
TIM00520
TIM00530
TIM00540
TIM00550
TIM00560
TIM00570
TIM00580
TIM00590
TIM03130
TIM03140
TIM03150
TIM00460
TIM00470
TIM00480
TIM03180
TIM03190
TIM00620

/*
THIS INSTRUCTION ENTERS THE BIT BY BIT PRODUCT OF
Y_BAR AND THE CONTENTS OF THE Q REGISTER INTO THE
A-REGISTER. THE PARITY OF THE A REGISTER IS CHECKED
BY THE CALL TO THE PROCEDURE (PAR_CHECK). IF THE PARITY
IS EVEN AND THE VALUE OF THE J DESIGNATOR (J_DEC)
IS 2 THEN SKIP THE NEXT INSTRUCTION OR IF THE
PARITY IS ODD AND THE VALUE OF THE J DESIGNATOR IS
3 THEN SKIP THE NEXT INSTRUCTION. NOTE THE
USE OF THE PL/I FUNCTION (BOOL).
*/
TIM03100
TIM03110
TIM03120
TIM00500
TIM00510
TIM00520
TIM00530
TIM00540
TIM00550
TIM00560
TIM00570
TIM00580
TIM00590
TIM03130
TIM03140
TIM03150
TIM00460
TIM00470
TIM00480
TIM03180
TIM03190
TIM00620

VAR (40): CALL READ IT;
A = BOOL (Y_BAR,Q,'0001');
PARITY = PAR_CHECK (A);
J_DEC = OCTAL (J);
IF J_DEC = 2 & PARITY = 'EVEN' THEN GO TO SKIP;
IF J_DEC = 3 & PARITY = 'ODD' THEN GO TO SKIP;
GO TO ENDING;
TIM03100
TIM03110
TIM03120
TIM00500
TIM00510
TIM00520
TIM00530
TIM00540
TIM00550
TIM00560
TIM00570
TIM00580
TIM00590
TIM03130
TIM03140
TIM03150
TIM00460
TIM00470
TIM00480
TIM03180
TIM03190
TIM00620

/*
THIS INSTRUCTION ADDS TO THE CONTENTS OF THE A

```


REGISTER THE BIT BY BIT PRODUCT OF THE OPERAND
Y_BAR AND THE CONTENTS OF THE Q REGISTER. NOTE THE
USE OF THE PL/1 FUNCTIONS (BIT,BOOL,BINARY).

```

VAR (41): CALL READ_IT;
          A = BIT (BINARY (A,15,0) +
                  BINARY (BOOL (Y_BAR,Q,'0001'),15,0),30);
          CALL J_CHANGE;
          GO TO ENDING;

```

/* THIS INSTRUCTION SUBTRACTS FROM THE CONTENTS OF
THE A REGISTER THE BIT BY BIT PRODUCT OF THE
OPERAND Y_BAR AND THE CONTENTS OF THE Q REGISTER.
NOTE THE USE OF THE PL/1 FUNCTIONS (BIT,BOOL,BINARY).

```

VAR (42): CALL READ_IT;
          A = BIT (BINARY (A,15,0) -
                  BINARY (BOOL (Y_BAR,Q,'0001'),15,0),30);
          CALL J_CHANGE;
          GO TO ENDING;

```

/* THIS INSTRUCTION SUBTRACTS FROM THE CONTENTS OF THE
A REGISTER THE BIT BY BIT PRODUCT OF THE OPERAND
Y_BAR AND THE CONTENTS OF THE Q REGISTER. THE ORIGINAL
CONTENTS OF THE A & Q REGISTERS ARE SAVED OFF IN THE
VARIABLES A1 AND Q1. THE CONTENTS OF THE ORIGINAL
REGISTERS ARE NOT DISTURBED BY THIS INSTRUCTION.
NOTE THE USE OF THE PL/1 FUNCTIONS (BIT,BOOL,BINARY).

```

VAR (43): CALL READ_IT;
          A1 = A;
          Q1 = Q;
          A = BIT (BINARY (A,15,0) -
                  BINARY (BOOL (Y_BAR,Q,'0001'),15,0),30);
          CALL J_CHANGE;
          A = A1;
          Q = Q1;
          GO TO ENDING;

```

/* THIS INSTRUCTION ENTERS INTO THE A REGISTER THE BIT
BIT PRODUCT OF THE OPERAND Y AND THE CONTENTS OF THE
Q REGISTER. THE PARITY OF THE A REGISTER IS THEN CHECKED
BY THE CALL TO THE PROCEDURE (PAR_CHECK). IF THE PARITY
EVEN AND THE VALUE OF THE J DESIGNATOR IS 2 OR THE
PARITY IS ODD AND THE VALUE OF THE J DESIGNATOR IS 3
THEN SKIP TO THE NEXT INSTRUCTION. OTHERWISE STORE THE
CONTENTS OF THE A REGISTER AT STORAGE ADDRESS Y AS
MODIFIED BY THE K DESIGNATOR. THE STORING IS ACCOMPLISHED

```

TIM00630
TIM00640
TIM00650
TIM00660
TIM00670
SIM03210
SIM03220
SIM03230
SIM03240
SIM03250
TIM00690
TIM00700
TIM00710
TIM00720
TIM00730
SIM03260
SIM03270
SIM03280
SIM03290
SIM03300
SIM03310
TIM00800
TIM00810
TIM00820
TIM00830
TIM00840
TIM00850
TIM00860
TIM00870
SIM03320
TIM00750
TIM00760
SIM03330
SIM03340
SIM03350
TIM00770
TIM00780
SIM03360
SIM03370
TIM00900
TIM00910
TIM00920
TIM00930
TIM00940
TIM00950
TIM00960
TIM00970
TIM00980

```



```

BY THE CALL TO THE PROCEDURE (REP_STO).
*/
TIM00990
TIM01000
SIM03380
SIM03390
TIM01010
TIM01020
TIM01030

VAR (44): A = BOOL (MEMORY (VAL_15 (Y)),Q,'0001');
PARITY = PAR_CHECK (A);
J_DEC = OCTAL (J);
IF J_DEC = 2 & PARITY = 'EVEN' THEN GO TO SKIP;
IF J_DEC = 3 & PARITY = 'ODD' THEN GO TO SKIP;
R_DEC = OCTAL (R);
IF R_DEC > 4 & FLAG = '1'B THEN
Y = BIT (BINARY (Y,15,0) + BINARY (B_REGIS (6),15,0),15);
CALL REP_STO (A,Y);
GO TO ENDING;

/*
THIS INSTRUCTION ADDS TO THE CONTENTS OF THE A
REGISTER THE BIT BY BIT PRODUCT OF THE OPERAND
Y AND THE CONTENTS OF THE Q REGISTER. THE CONTENTS
OF THE A REGISTER ARE THEN STORED AT STORAGE
ADDRESS Y AS MODIFIED BY THE K DESIGNATOR IN
THE CALL TO THE PROCEDURE (REP_STO).
*/

VAR (45): A = BIT (BINARY (A,15,0) +
BINARY (BOOL (Y,Q,'0001'),15,0),30);
R_DEC = OCTAL (R);
IF R_DEC > 4 & FLAG = '1'B THEN
Y = BIT (BINARY (Y,15,0) + BINARY (B_REGIS (6),15,0),15);
CALL REP_STO (A,Y);
CALL J_CHANGE;
GO TO ENDING;

/*
THIS INSTRUCTION SUBTRACTS FROM THE CONTENTS OF
THE A REGISTER THE BIT BY BIT PRODUCT OF THE
OPERAND Y AND THE CONTENTS OF THE Q REGISTER. THE
RESULT IS THEN STORED AT STORAGE ADDRESS Y AS
MODIFIED BY THE K DESIGNATOR IN THE CALL TO
THE PROCEDURE (REP_STO).
*/

VAR (46): A = BIT ( BINARY (A,15,0) -
BINARY (BOOL (MEMORY (VAL_15 (Y)),Q,'0001'),15,0),30);
R_DEC = OCTAL (R);
IF R_DEC > 4 & FLAG = '1'B THEN
Y = BIT (BINARY (Y,15,0) + BINARY (B_REGIS (6),15,0),15);
CALL REP_STO (A,Y);
CALL J_CHANGE;
GO TO ENDING;

/*
THIS INSTRUCTION STORES AT ADDRESS Y AS MODIFIED
BY THE K DESIGNATOR IN THE CALL TO PROCEDURE
(REP_STO) THE BIT BY BIT PRODUCT OF THE
TIM03420
SIM03430
SIM03440
TIM01070
TIM01080
TIM01090
TIM01100
TIM01110
TIM01120
TIM01130
SIM03450
SIM03460

TIM03470
SIM03480
SIM03490
SIM03500
TIM01160
TIM01170
TIM01180
TIM01190
TIM01200
TIM01210
TIM01220
SIM03510
SIM03520

TIM03530
SIM03540
SIM03550
SIM03560
TIM01260
TIM01270
TIM01280

```



```

CONTENTS OF THE A AND Q REGISTERS.          */

VAR (47): CALL STORE (BOOL (A,Q,'0001'),Y);
        CALL J_CHANGE;    GO TO ENDING;

/* THIS INSTRUCTION SETS THE BITS OF A TO 1 BITS OF
   OPERAND Y VIA PL/1 FUNCTION BOOL          */

VAR (50): CALL READ_IT;    A = BOOL (A,Y_BAR,'0111');
        CALL J_CHANGE;    GO TO ENDING;

/* THIS INSTRUCTION COMPLIMENTS THE BITS OF A TO 1 BITS
   CORRESPONDING TO 1 BITS IN Y VIA PL/1 FUNCTION BOOL */

VAR (51): CALL READ_IT;    A = BOOL (A,Y_BAR,'0100');
        CALL J_CHANGE;    GO TO ENDING;

/* THIS INSTRUCTION SETS THE BITS OF A TO 0 CORRESPONDING
   TO THE 1 BITS IN Y VIA THE PL/1 FUNCTION BOOL          */

VAR (52): CALL READ_IT;    A = BOOL (A,Y_BAR,'0010');
        CALL J_CHANGE;    GO TO ENDING;

/* THIS INSTRUCTION SETS THE BITS OF A TO BITS OF Y
   CORRESPONDING TO THE 1 BITS OF Q VIA THE PL/1 BOOL          */

VAR (53): CALL READ_IT;
        A = BOOL (A,BOOL (Y_BAR,Q,'0001'),'0111');
        CALL J_CHANGE;    GO TO ENDING;

/* THIS INSTRUCTION SETS THE INDIVIDUAL BITS OF A TO 1
   CORRESPONDING TO THE U 1 BITS OF THE OPERAND Y.
   IT THEN STORES THE CONTENTS OF REGISTER A INTO
   LOCATION Y. THE CALL TO REP STO ACCOMPLISHES THE STORING */

VAR (54): A = BOOL (A,MEMORY (VAL_15 (Y)),'0111');

```



```

SIM03820
CALL J_CHANGE;
R_DEC = OCTAL (R);
IF R_DEC > 4 & FLAG = '1'B THEN
Y = BIT (BINARY (Y,15,0) + BINARY (B_REGIS (6),15,0),15);
CALL REP_STO (A,Y);
GO TO ENDING;

/*
THIS INSTRUCTION COMPLIMENTS THE BITS OF A CORRESPONDING
TO THE 1 BITS OF THE OPERAND Y AND THEN STORES THE
CONTENTS OF A AT LOCATION Y VIA THE CALL TO REPSTO. */
VAR (55): A = BOOL (A, MEMORY (VAL_15 (Y)), '0100');
CALL J_CHANGE;
R_DEC = OCTAL (R);
IF R_DEC > 4 & FLAG = '1'B THEN
Y = BIT (BINARY (Y,15,0) + BINARY (B_REGIS (6),15,0),15);
CALL REP_STO (A,Y);
GO TO ENDING;

/*
THIS INSTRUCTION SETS THE BITS OF A TO 0 CORRESPONDING
TO THE 1 BITS OF THE OPERAND Y AND THEN STORES THE
CONTENTS OF A AT THE LOCATION Y VIS THE CALL TO REP STO. */
VAR (56): A = BOOL (A, MEMORY (VAL_15 (Y)), '0010');
CALL J_CHANGE;
R_DEC = OCTAL (R);
IF R_DEC > 4 & FLAG = '1'B THEN
Y = BIT (BINARY (Y,15,0) + BINARY (B_REGIS (6),15,0),15);
CALL REP_STO (A,Y);
GO TO ENDING;

/*
THIS INSTRUCTION SETS THE BITS OF A TO 0 CRRRESPONDING
TO THE 1 BITS OF Q. IT THEN FORMS THE BIT BY BIT
PRODUCT OF Q AND THE OPERAND Y AND SETS THE BITS OF A
TO 1 CORRESPONDING TO THE 1 BUTS OF THE PRODUCT. IT
THEN STORES THE CONTENTS OF A AT LOCATION Y, */
VAR (57): A = BOOL (A,Q,'0010');
A = BOOL (A,BOOL (MEMORY (VAL_15 (Y)),Q,'0001'), '0111');
CALL J_CHANGE;
R_DEC = OCTAL (R);
IF R_DEC > 4 & FLAG = '1'B THEN
Y = BIT (BINARY (Y,15,0) + BINARY (B_REGIS (6),15,0),15);
CALL REP_STO (A,Y);
GO TO ENDING;

/*
THIS INSTRUCTION IS AN ARITHMETIC JUMP TO THE
K MODIFIED Y OPERAND ADDRESS AS DETERMINED BU THE J
DESIGNATOR FOR VARIOUS ARITHMETIC COMPARISONS OF THE

```


*/

A AND Q REGISTERS.

```

VAR (60): CALL READ_IT;
J_DEC = OCTAL (J);
IF J_DEC = 0 THEN GO TO ENDING;
IF J_DEC = 1 & Q > 0 |
J_DEC = 2 & Q < 0 |
J_DEC = 3 & A = 0 |
J_DEC = 4 & A = 0 |
J_DEC = 5 & A > 0 |
J_DEC = 6 & A < 0 |
J_DEC = 7 THEN DO;
P = SUBSTR (Y_BAR,16);
P_OCT = VAL_15 (P);
P_DEC = P;
GO TO START2;
END;
GO TO ENDING;

```

/* THIS INSTRUCTION JUMPS TO THE K MODIFIED Y OPERAND ADDRESS FOR THE SPECIAL CONDITIONS OF THE J DESIGNATOR AND THE JUMP AND STOP MACHINE CONSOLE SWITCHES. THESE SWITCHES ARE SIMULATED AND ARE SET TO 0 INITIALLY.

*/

```

VAR (61): CALL READ_IT;
J_DEC = OCTAL (J);
IF J_DEC = 0 & JUMP1 |
J_DEC = 1 & JUMP2 |
J_DEC = 2 & JUMP3 |
J_DEC = 3 & JUMP3 THEN DO;
P = SUBSTR (Y_BAR,16);
P_OCT = VAL_15 (P);
P_DEC = P;
END;
IF J_DEC = 4 & STOP5 |
J_DEC = 5 & STOP6 |
J_DEC = 6 & STOP7 |
J_DEC = 7 & STOP7 THEN GO TO FINISH;
GO TO ENDING;

```

/* THIS INSTRUCTION IS DESIGNED TO JUMP TO A NEW PROGRAM ADDRESS AS DETERMINED BY THE J HAT INPUT CHANNEL AND THE INPUT BUFFER. IT IS TOTALLY IGNORED IN THIS TRANSLATION PACKAGE.

*/

SIM04020
LES00040
LES00050
LES00060
LES00070
LES00080
LES00090
LES00100
LES00110
LES00120
LES00130
LES00140
LES00150
LES00160
LES00170
LES00180
SIM04030
SIM04040

SIM04050
LES00220
LES00230
LES00240
LES00250
LES00260
LES00270
LES00280
LES00290
LES00300
LES00310
LES00320
LES00330
LES00340
LES00350
LES00360
LES00370
SIM04060
SIM04070


```

SIM04080
SIM04090
SIM04100

VAR (62): CALL READ IT;
GO TO ENDING;

/* THIS INSTRUCTION IS DESIGNED TO JUMP TO A NEW PROGRAM
ADDRESS AS DETERMINED BY THE ACTIVITY OF THE OUTPUT
CHANNEL DESIGNATED BY J HAT. THIS INSTRUCTION
IS TOTALLY IGNORED IN THIS TRANSLATION.

*/

VAR (63): CALL READ IT;
GO TO ENDING;

/* THIS INSTRUCTION EXECUTES A JUMP TO THE OPERAND Y + 1
FOR J DESIGNATED COMPARISONS WITH THE A AND Q REGISTERS.
THE PRESENT INSTRUCTION ADDRESS IS SAVED IN ADDRESS
OF THE K MODIFIED OPERAND Y;

*/

VAR (64): CALL READ IT;
J_DEC = OCTAL (J);
IF J_DEC = 0 THEN GO TO ENDING;
IF J_DEC = 1
  J_DEC = 2 & Q > 0 |
  J_DEC = 3 & Q < 0 |
  J_DEC = 4 & A = 0 |
  J_DEC = 5 & A = 0 |
  J_DEC = 6 & A > 0 |
  J_DEC = 7 & A < 0 | THEN DO;
HOLD4 = VAL THIRTY (Y_BAR);
SUBSTR ( MEM2 (HOLD4 - 32767), 16, 15) = P;
ELSE SUBSTR (MEM1 (HOLD4), 16, 15) = P;
P = BIT (HOLD4, 15);
END; GO TO ENDING;

/* THIS INSTRUCTION IS DESIGNED TO JUMP TO A NEW PROGRAM
ADDRESS BASED UPON THE J DESIGNATOR AND THE CONSOLE
SWITCH SETTINGS. THIS INSTRUCTION IS IGNORED.

*/

SIM04150
SIM04160

VAR (65): CALL READ IT;
GO TO ENDING;

/* THIS INSTRUCTION IS DESIGNED TO MANIPULATE INPUT BUFFERS,
CHANNELS, ETC. ON THE CP 901 AND IS NOT APPLICABLE
FOR THIS TRANSLATION.

*/

SIM04170
SIM04180
SIM04190

VAR (66): PUT FILE (SYSPRINT) EDIT ('INTERRUPT BUFFER') (SKIP, A);
GO TO ENDING;

SIM04210

```



```

/*
THIS INSTRUCTION IS DESIGNED TO MANIPULATE OUTPUT
BUFFERS AND CHANNELS ON THE CP 901 AND IS NOT
APPLICABLE FOR THIS TRANSLATION.
*/

```

```

VAR (67):PUT FILE (SYSPRINT) EDIT ('VAR LABEL 67') (SKIP,A);
GO TO ENDING;

```

```

/*
THIS INSTRUCTION SETS B REGISTER # 7 TO THE
LOWER FIFTEEN BITS OF THE K MODIFIED OPERAND Y.
IF THE LOWER 15 BITS OF Y ARE NON ZERO THEN REPEAT THE
NEXT INSTRUCTION FOR THE COUNTER IN THE LOWER 15
BITS OF Y. THE J DESIGNATOR IS NOW THE R DESIGNATOR.
IT HAS THE FOLLOWING SPECIAL MODS:
R = 0 OPERAND IS CONSTANT
R = 1 OPERAND INCREMENTED BY + 1 EACH EXECUTION
R = 2 OPERAND DECREASED BY 1 EACH EXECUTION;
R = 3 REPEAT INITIAL B REGISTER MODIFICATION
FOR EACH EXECUTION
R = 4 MODIFY ONLY REPLACE INSTRUCTION BY B
REGISTER (6) FOR STORING
R = 5 ADD 1 TO OPERAND ADDRESS MODIFY REPLACE
INSTRUCTION BY B REGISTER (6) FOR STORING
R = 6 SUBTRACT 1 FROM OPERAND ADDRESS MODIFY
REPLACE INSTRUCTIONS BY B REGISTER (6) FOR STORING
R = 7 REPEAT ORIGINAL B REGISTER MODIFICATION
AND MODIFY REPLACE INSTRUCTIONS BY B REGISTER
(6) FOR STORING;
REPLACE INSTRUCTIONS ARE : 24,25,34-37,44-46,54-57.
*/

```

```

VAR (70): CALL READ_IT;
IF Y_BAR = 0 THEN DO;
P_DEC = P_DEC + 1;
P = BIT (P_DEC, 15);
END;
ELSE DO;
B_REGIS (7) = SUBSTR (Y_BAR,16,15);
R = J;
R_DEC = OCTAL (R);
HOLD5 = VAL_15 (B_REGIS (7));
HOLD2 = P_DEC + 1;
HOLD1 = 0;
P_DEC = HOLD2;
HOLD1 = HOLD1 + 1;
DO L = HOLD1 TO HOLD5;
FLAG = '1'B;
P = BIT (P_DEC,15);
GO TO START2;

```



```

END;
END;
GO TO ENDING;

/*
THIS INSTRUCTION SKIPS THE NEXT INSTRUCTION IF THE
B REGISTER DESIGNATED BY J IS EQUAL TO THE LOWER 15
BITS OF THE K MODIFIED Y OPERAND. IN THIS CASE THAT
B REGISTER IS ZEROED. ELSE EXECUTE THE NEXT INSTRUCTION
AND INCREMENT THE DESIGNATED B REGISTER BY 1.
*/

VAR (71): CALL READ_IT;
J_DEC = OCTAL (J);
IF B_REGIS (J_DEC) = SUBSTR (Y_BAR,16,15) THEN DO;
    B_REGIS (J_DEC) = 0;
    P_DEC = P_DEC + 1;
    P = BIT (P_DEC, 15);
END;
ELSE B_REGIS (J_DEC) = BIT (B_REGIS (J_DEC),15,0) +
    1,15);
GO TO ENDING;

/*
THIS INSTRUCTION EXECUTES A JUMP TO THE K MODIFIED
OPERAND Y ADDRESS IF THE J DESIGNATED B REGISTER IS
NOT ZERO. (REDUCE IT BY 1 IN THIS CASE).
*/

VAR (72): CALL READ_IT;
IF BINARY (B_REGIS (J),15,0) = 0 THEN DO;
    B_REGIS (J) = BIT (BINARY (B_REGIS (J),15,0) -
        1,15);
    P = VAL 15 (SUBSTR (Y_BAR,16,15));
    GO TO START2;
END;
GO TO ENDING;

/*
THIS INSTRUCTION ESTABLISHES AN INPUT BUFFER AS
DETERMINED BY THE K HAT DESIGNATOR MODIFYING THE
Y OPERAND. ADDRESS 100 PLUS J HAT (HOLD3) CONTAINS
THE SCOPE OF THE AREA FOR THE READING. READING WILL
CONTINUE UNTIL THE UPPER HALF OF THE LOCATION (HOLD3)
AND THE LOWER HALF ARE EQUAL. IF K HAT = 0 Y IS
STORED IN THE LOWER 15 BITS OF THE LOCATION (HOLD3).
IF K HAT = 1 THE LOWER 15 BITS OF THE LOCATION OF (
Y ARE STORED INTO THE LOWER 15 BITS OF (HOLD3).
K HAT = 2 IS NOT ALLOWED, IF K HAT = 3 LOCATION
(HOLD3) IS SET TO THE LOCATION OF THE OPERAND Y.
*/

VAR (73): PUT FILE (SYSPRINT) EDIT ('VAR LABEL 73') (SKIP,A);
K_DEC = OCTAL ('0'B || K_HAT);

```

SIM04280
SIM04290

SIM04300

SIM04310
SIM04320

SIM04340
SIM04350
SIM04360

SIM04380
SIM04390
SIM04400
SIM04410
SIM04420


```

ASTR = MEMORY (VAL_15 (Y));
J_DEC = OCTAL ('00'B || SUBSTR (J_HAT,1,1)) * 10 +
OCTAL (SUBSTR (J_HAT,2,3));
HOLD3 = 100 + J_DEC;
SUBSTR (MEM1 (HOLD3),16,15) = Y;
IF K_DEC = 1 THEN
SUBSTR (MEM1 (HOLD3),16,15) = SUBSTR (ASTR,16,15);
IF K_DEC = 3 THEN
MEM1 (HOLD3) = ASTR;
DO WHILE (SUBSTR (MEM1 (HOLD3),1,15) !=
SUBSTR (MEM1 (HOLD3),16,15));
ON ENDFILE (SYSIN) BEGIN;
PUT FILE (SYSPRINT) EDIT ('EXPECTED MORE INPUT ') (SKIP,A);
GO TO ENDING;
END;
GET FILE (SYSIN) EDIT (INPUT1) (A(30));
MEM1 (HOLD3) = BIT (BINARY (MEM1 (HOLD3)) + 1);
END;
GO TO ENDING;

```

SIM04440
SIM04450

```

/*
THIS INSTRUCTION ESTABLISHES AN OUTPUT BUFFER USING
AN AREA DEFINED BY THE K HAT MODIFIED Y OPERAND.
IF K HAT = 2 THIS IS AN EXTERNAL FUNCTION BUFFER AND
IS NOT IMPLEMENTED IN THIS TRANSLATOR. J HAT
DESIGNATES THE DATA CHANNEL AND IS ALSO NOT IMPLEMENTED.
ADDRESS 120 PLUS J HAT IS THE CONTROL ADDRESS (HOLD3).
IF K HAT = 0 THEN THE LOWER 15 BITS OF LOCATION
(HOLD3) = Y; THEN THE LOWER 15 BITS OF LOCATION
(HOLD3) = LOWER 15 BITS OF LOCATION (Y). IF K HAT
= 3 THEN THE LOCATION (HOLD3) = LOCATION IN (Y).
OUTPUT WILL CONTINUE STARTING AT THE INITIAL ADDRESS
OF THE LOWER 15 BITS OF LOCATION (HOLD3) UNTIL THE
UPPER 15 BITS = LOWER 15 BITS.
*/

```

```

VAR (74): PUT FILE (SYSPRINT) EDIT ('VAR LABEL 74') (SKIP,A);
K_DEC = OCTAL ('00'B || K_HAT);
J_DEC = OCTAL ('00'B || SUBSTR (J_HAT,1,1)) * 10 +
OCTAL (SUBSTR (J_HAT,2,3));
HOLD3 = 120 + J_DEC;
ASTR = MEMORY (VAL_15 (Y));
IF K_DEC = 0 THEN
SUBSTR (MEM1 (HOLD3),16,15) = Y;
IF K_DEC = 1 THEN
SUBSTR (MEM1 (HOLD3),16,15) = SUBSTR (ASTR,16,15);
IF K_DEC = 3 THEN
MEM1 (HOLD3) = ASTR;
DO WHILE (SUBSTR (MEM1 (HOLD3),1,15) != SUBSTR (MEM1 (HOLD3),

```



```

16,15));
HOLD5 = VAL_15 (SUBSTR (MEM1 (HOLD3),16,15));
PUT FILE (SYSPRINT) EDIT ('MEMORY ', HOLD5, ' IS ',
      MEMORY (HOLD5)) (SKIP,A,F(8),A,B);
SUBSTR (MEM1 (HOLD3),16,15) = BIT (BINARY (SUBSTR (MEM1 (HOLD3),
16,15),15,0) + 1,15);
END;
GO TO ENDING;
SIM04470
SIM04480

/*      THIS INSTRUCTION ESTABLISHES AN INPUT BUFFER WITH
MONITOR AND IS THEREFORE NOT IMPLEMENTED.      */

VAR (75):PUT FILE (SYSPRINT) EDIT ('VAR LABEL 75') (SKIP,A);
GO TO ENDING;
SIM04500
SIM04510

/*      THIS INSTRUCTION ESTABLISHES AN OUTPUT BUFFER WITH
MONITOR AND IS THEREFORE NOT IMPLEMENTED.      */

VAR (76):PUT FILE (SYSPRINT) EDIT ('VAR LABEL 76') (SKIP,A);
GO TO ENDING;
SIM04530
SIM04540

/*      THIS INSTRUCTION IS THE SERIES OF THE DIRECT
ADDRESSING MODE FUNCTION. VARIABLE FS NOW HAS THE
OPCODE FOR THE FOLLOWING FUNCTIONS.      */

VAR (77):PUT FILE (SYSPRINT) EDIT ('VAR LABEL 77') (SKIP,A);
F_DEC = OCTAL (SUBSTR (FS,1,3)) * 10 +
      OCTAL (SUBSTR (FS,4,3));
IF F_DEC = 11 THEN GO TO VAR_11;
IF F_DEC = 15 THEN GO TO VAR_15;
IF F_DEC = 44 THEN GO TO VAR_44;
IF F_DEC = 50 THEN GO TO VAR_50;
IF F_DEC = 54 THEN GO TO VAR_54;
IF F_DEC = 63 THEN GO TO VAR_63;
IF F_DEC = 64 THEN GO TO VAR_64;
IF F_DEC = 65 THEN GO TO VAR_65;
GO TO ENDING;
SIM04560
SIM04570

/*      THIS INSTRUCTION LOADS THE A REGISTER WITH THE MEMORY
ADDRESS OF THE LEAST SIGNIFICANT 17 BITS OF THE U (Y2)      */

VAR_11:  HOLD4 = VAL_15 (SUBSTR (Y_2,3));
HOLD4 = HOLD4 + (1000000 * OCTAL ('0'B || SUBSTR (Y_2,1,2)));
A = MEMORY (HOLD4);
GO TO ENDING;

/*      THIS INSTRUCTION IS A DIRECT ADDRESSING INSTRUCTION
AND STORES THE CONTENTS OF THE A REGISTER IB LOCATION

```



```

DETERMINED BY THE LEAST SIGNIFICANT 17 BITS OF THE
U REGISTER (Y_2).
*/

VAR_15:  HOLD4 = VAL_15 (SUBSTR (Y_2,3));
        HOLD4 = HOLD4 + (1000000 * OCTAL ('0'B || SUBSTR (Y_2,1,2)));
        IF HOLD4 > 32767 THEN
            MEM2 (HOLD4 - 32767) = A;
        ELSE MEM1 (HOLD4) = A;
        GO TO ENDING;

/*      THIS INSTRUCTION TESTS THE MODIFIED Y OPERAND FOR 0
        IF IT IS 0 THEN NEXT INSTRUCTION IS SKIPPED.
*/

VAR_44:  IF Y = 0 THEN DO;
        P_DEC = P_DEC + 1;
        P = BIT (P_DEC,15);
        END;
        GO TO ENDING;

/*      THIS INSTRUCTION IS DIRECT ADDRESSING AND STORES THE
        SEVEN LEAST SIGNIFICANT BITS OF K HAT MODIFIED Y OPERAND
        INTO ABSOLUTE PAGE REGISTER DESIGNATED BY J HAT AND IS
        THUS NOT IMPLIMENTED.
*/

VAR_50:  GO TO ENDING;
/*      THIS INSTRUCTION IN DIRECT ADDRESSING MODE STORES
        BITS OF ABSOLUTE PAGE REGISTER DESIGNATED BY J HAT
        INTO THE K HAT MODIFIED Y OPERAND. IT IS NOT IMPLEMENTED.
*/

VAR_54:  GO TO ENDING;

/*      THIS DIRECT ADDRESSING INSTRUCTION LOADS THE VALUE OF THE
        P REGISTER INTO THE B 7 REGISTER AND JUMPS TO LOWER 15
        BITS OF THE Y MODIFIED OPERAND TO SAVE PROGRAMS IN SHARED
        MEMORY. (IT IS NOT IMPLEMENTED).
*/

VAR_63:  GO TO ENDING;
/*      SAME AS 77 63 BUT DIRECT MODE SET
*/

VAR_64:  GO TO ENDING;
/*      SAME AS 77 63 BUT WITH PAGING MODE SET
*/

VAR_65:  GO TO ENDING;
ENDING:  IF FLAG = '1'B THEN DO;
        FLAG = '0'B;

```



```

GO TO REPEAT;
END;
P_DEC = P_DEC + 1;
P = BIT (P_DEC, 15);
GO TO START2;

OCTAL: PROCEDURE (INP) FIXED BINARY;
DCL INP = '000'; THEN RETURN (0);
IF INP = '001'; THEN RETURN (1);
IF INP = '010'; THEN RETURN (2);
IF INP = '011'; THEN RETURN (3);
IF INP = '100'; THEN RETURN (4);
IF INP = '101'; THEN RETURN (5);
IF INP = '110'; THEN RETURN (6);
RETURN (7);
END OCTAL;

REP_STR: PROCEDURE (ABIT, ANUM);
PUT-FILE(SYSPRINT) EDIT ('IN PROC REP_STR ' ) (SKIP, A);
DCL ABIT BIT (1);
DCL ANUM FIXED BINARY (15);
ASTR = ' ';
OVER: IF ANUM = 0 THEN GO TO STOPIT;
ELSE DO;
ASTR = ASTR || ABIT;
ANUM = ANUM - 1;
GO TO OVER;
END;
STOPIT: END REP_STR;

VAL_THIRTY: PROCEDURE (REGIS) FIXED BINARY (15);
PUT-FILE(SYSPRINT) EDIT ('IN PROC VAL_THIRTY' ) (SKIP, A);
DCL REGIS BIT (30);
RETURN (OCTAL (SUBSTR (REGIS, 1, 3)) * (10 ** 9) +
OCTAL (SUBSTR (REGIS, 4, 3)) * (10 ** 8) +
OCTAL (SUBSTR (REGIS, 7, 3)) * (10 ** 7) +
OCTAL (SUBSTR (REGIS, 10, 3)) * 1000000 +
OCTAL (SUBSTR (REGIS, 13, 3)) * 100000 +
OCTAL (SUBSTR (REGIS, 16, 3)) * 10000 +
OCTAL (SUBSTR (REGIS, 19, 3)) * 1000 +
OCTAL (SUBSTR (REGIS, 22, 3)) * 100 +
OCTAL (SUBSTR (REGIS, 25, 3)) * 10 +
OCTAL (SUBSTR (REGIS, 28, 3)) );

END VAL_THIRTY;

VAL_15: PROCEDURE (REGIS) FIXED BINARY (15);
PUT-FILE(SYSPRINT) EDIT ('IN PROC VAL_15 ' ) (SKIP, A);

```

SIM04590
SIM04600
SIM04610
SIM04620
SIM04630
SIM04640
SIM04650
SIM04660
SIM04670
SIM04680
SIM04690
SIM04700
SIM04710
SIM04720
SIM04730
SIM04740

SIM04760
SIM04770
SIM04780
SIM04790
SIM04800
SIM04810
SIM04820
SIM04830
SIM04840
SIM04850
SIM04860
SIM04870

SIM04880
SIM04890
SIM04900
SIM04910
SIM04920
SIM04930
SIM04940
SIM04950
SIM04960
SIM04970
SIM04980
SIM04990
SIM05000
SIM05010


```

DCL REGIS BIT (15);
RETURN (OCTAL (SUBSTR (REGIS,1,3)) * 10000 +
        OCTAL (SUBSTR (REGIS,4,3)) * 1000 +
        OCTAL (SUBSTR (REGIS,7,3)) * 100 +
        OCTAL (SUBSTR (REGIS,10,3)) * 10
        OCTAL (SUBSTR (REGIS,13,3)) );

END VAL_15;

CIR_SFT: PROCEDURE (STRNG,VALUE) BIT (30);
PUT FILE(SYSPRINT) EDIT (,'IN PROC CIR SFT ')(SKIP,A);
DCL STRNG BIT (30);
DCL VALUE FIXED BIN (15);
CS: IF VALUE = 0 THEN RETURN (STRNG);
    ELSE DO;
        STRNG = SUBSTR (STRNG,2,29) || SUBSTR (STRNG,1,1);
        VALUE = VALUE - 1;
        GO TO CS;
    END;
END CIR_SFT;

STORE: PROCEDURE (BITS,BATS);
PUT FILE(SYSPRINT) EDIT (,'IN PROC STORE ')(SKIP,A);
DCL BITS BIT (30);
DCL BATS BIT (15);
HOLD4 = VAL_15 (BATS);
IF K = 0 THEN Q = BITS;
IF K = 1 THEN IF HOLD4 > 32767 THEN MEM2 (HOLD4 - 13516) =
    SUBSTR (MEM2 (HOLD4 - 13516),1,15) ||
    SUBSTR (BITS,16,15);
    ELSE MEM1 (HOLD4) = SUBSTR (MEM1 (HOLD4),1,15) ||
    SUBSTR (BITS,16,15);
    IF K = 2 THEN IF HOLD4 > 32767 THEN MEM2 (HOLD4 - 13516) =
    SUBSTR (BITS,16,15) ||
    SUBSTR (MEM2 (HOLD4 - 13516),1,15);
    ELSE MEM1 (HOLD4) = SUBSTR (MEM1 (HOLD4),1,15);
    IF K = 3 THEN IF HOLD4 > 32767 THEN MEM2 (HOLD4 - 13516) =
    SUBSTR (MEM1 (HOLD4) = BITS;
    IF K = 4 THEN A = BITS;
    IF K = 5 THEN IF HOLD4 > 32767 THEN
    SUBSTR (MEM2 (HOLD4 - 32767),16,15) = ¬ (BITS);
    ELSE SUBSTR (MEM1 (HOLD4),16,15) = ¬ (BITS);
    IF K = 6 THEN IF HOLD4 > 32767 THEN
    SUBSTR (MEM2 (HOLD4 - 32767),1,15) = ¬ (BITS);
    ELSE SUBSTR (MEM1 (HOLD4),1,15) = ¬ (BITS);
    IF K = 7 THEN IF HOLD4 > 32767 THEN
    MEM2 (HOLD4 - 32767) = ¬ (BITS);
    ELSE MEM1 (HOLD4) = ¬ (BITS);

```

SIM05020
SIM05030
SIM05040
SIM05050
SIM05060
SIM05070
SIM05080
SIM05090
SIM05100

SIM05120
SIM05130
SIM05140
SIM05150
SIM05160
SIM05170
SIM05180
SIM05190
SIM05200
SIM05210
SIM05220

SIM05230
SIM05240
SIM05250
SIM05260
SIM05270
SIM05280
SIM05290
SIM05300
SIM05310
SIM05320
SIM05330
SIM05340
SIM05350
SIM05360
SIM05370
SIM05380
SIM05390


```

END STORE;

READ_IT: PROCEDURE;
PUT_FILE(SYSPRINT) EDIT ('IN PROC S READ IT ')(SKIP,A);
DCL VAL_K FIXED BINARY (15);
VAL_K = OCTAL(K);
CALL REP_STR ('0',15);
IF VAL_K = (0) THEN Y_BAR = ASTR || Y;
IF VAL_K = (1) THEN Y_BAR = ASTR || Y;

IF VAL_K = 2 THEN Y_BAR = ASTR || Y;
IF VAL_K = 3 THEN Y_BAR = ASTR || Y;
CALL REP_STR (SUBSTR (Y_BAR,16,1),15);
IF VAL_K = 4 THEN Y_BAR = ASTR || Y;
IF VAL_K = 5 THEN Y_BAR = ASTR || Y;

IF VAL_K = 6 THEN DO;
CALL REP_STR (SUBSTR (Y_BAR,1,1),15);
Y_BAR = ASTR || SUBSTR (MEMORY (VAL_15 (Y)),1,15);
END;
IF DEC = 22 | F_DEC = 52 | F_DEC = 53 THEN GO TO OMT;
IF VAL_K = 7 THEN Y_BAR = MEMORY (VAL_THIRTY (A));
PUT_FILE(SYSPRINT) EDIT ('YBAR AFTER READIT = ',Y_BAR) (SKIP,A,B(30));
OMT: END READ_IT;

REV_OCTAL: PROCEDURE (BIN) BIT(3);
PUT_FILE(SYSPRINT) EDIT ('IN PROC REV OCTAL ')(SKIP,A);
DCL BIN CHAR (1);
IF BIN = '0' THEN RETURN ('000'B);
IF BIN = '1' THEN RETURN ('001'B);
IF BIN = '2' THEN RETURN ('010'B);
IF BIN = '3' THEN RETURN ('011'B);
IF BIN = '4' THEN RETURN ('100'B);
IF BIN = '5' THEN RETURN ('101'B);
IF BIN = '6' THEN RETURN ('110'B);
IF BIN = '7' THEN RETURN ('111'B);

END REV_OCTAL;

REP-STO: PROCEDURE (EINS, ZWEI);
PUT_FILE(SYSPRINT) EDIT ('IN PROC REP STO ')(SKIP,A);
DCL EINS BIT (30);
DCL ZWEI BIT (15);
DCL VALUE FIXED BINARY (15);
IF (K = 0 | K = 4 | K = 7) THEN RETURN;
VALUE = VAL_K (ZWEI);
IF K = 1 | K = 5 THEN IF VALUE > 32767 THEN

```

SIM05400
SIM05410
SIM05420

SIM05430
SIM05440
SIM05450
SIM05460
SIM05470
SIM05480
SIM05490
SIM05500
SIM05510
SIM05520
SIM05530
SIM05540
SIM05550
SIM05560
SIM05570
SIM05580
SIM05590
SIM05600
SIM05610
SIM05620
SIM05630

SIM05640
SIM05650
SIM05660

SIM05670
SIM05680
SIM05690
SIM05700
SIM05710
SIM05720
SIM05730
SIM05740
SIM05750
SIM05760
SIM05770
SIM05780
SIM05790

SIM05800
SIM05810
SIM05820
SIM05830
SIM05840


```

SUBSTR (MEM2 (VALUE - 32767),16,15) =
SUBSTR (EINS,16,15);
ELSE SUBSTR (MEM1 (VALUE),16,15) = SUBSTR (EINS,16,15);
IF K = 2 THEN IF VALUE > 32767 THEN
SUBSTR (MEM2 (VALUE - 32767),1,15) =
SUBSTR (EINS,16,15);
ELSE SUBSTR (MEM1 (VALUE),1,15) = SUBSTR (EINS,16,15);
IF K = 3 THEN IF VALUE > 32767 THEN
MEM2 (VALUE - 32767) = EINS;
ELSE MEM1 (VALUE) = EINS;
END REP_STO;

```

SIM05860
SIM05870
SIM05880

```

REP_READ: PROCEDURE;
PUT_FILE(SYSPRINT) EDIT (IN PROC REP READ ')(SKIP,A);
CALL REP_STR (K);
IF K_Y_BAR = 1 THEN
IF K_DEC = 2 THEN SUBSTR (MEMORY (VAL_15 (Y)),16,15);
IF K_Y_BAR = 3 THEN
IF K_DEC = 3 THEN SUBSTR (MEMORY (VAL_15 (Y)),1,15);
CALL REP_STR (SUBSTR (VAL_15 (Y)));
IF K_DEC = 5 THEN SUBSTR (Y_BAR,17,1),15);
Y_BAR = ASTR (SUBSTR (MEMORY (VAL_15 (Y)),16,15);
CALL REP_STR (SUBSTR (Y_BAR,1,1),15);
IF K_DEC = 6 THEN
Y_BAR = ASTR (SUBSTR (MEMORY (VAL_15 (Y)),1,15);
END REP_READ;

```

SIM05900
SIM05910
SIM05920

```

J_CHANGE: PROCEDURE;
PUT_FILE(SYSPRINT) EDIT (IN PROC J CHANGE ')(SKIP,A);
CALL REP_STR (J);
IF J_DEC = 0 THEN RETURN;
IF J_DEC = 1 THEN GO TO SKIP2;
IF J_DEC = 2 THEN IF Q > -1 THEN GO TO SKIP2;
ELSE RETURN;
IF J_DEC = 3 THEN IF Q < 0 THEN GO TO SKIP2;
ELSE RETURN;
IF J_DEC = 4 THEN IF A = 0 THEN GO TO SKIP2;
ELSE RETURN;
IF J_DEC = 5 THEN IF A = 0 THEN GO TO SKIP2;
ELSE RETURN;
IF J_DEC = 6 THEN IF A > -1 THEN GO TO SKIP2;
ELSE RETURN;
IF J_DEC = 7 THEN IF A < 0 THEN GO TO SKIP2;
ELSE RETURN;
SKIP2: P_DEC = P_DEC + 1;

```

SIM05940
SIM05950
SIM05960
SIM05970
SIM05980
SIM05990
SIM06000
SIM06010
SIM06020
SIM06030
SIM06040
SIM06050
SIM06060
SIM06070
SIM06080
SIM06090


```

        P = BIT (P_DEC,15);
        END J_CHANGE;

SPEC_J: PROCEDURE;
DCL J_DEC FIXED BINARY (15);
J_DEC = OCTAL (J);
IF J_DEC = 1
    & A > 0
    & A < 0
    & Q = 0
    & Q > 0
    & Q < 0
    & Q = 0
    & Q > 0
    & Q < 0
    THEN DO;
        P_DEC = P_DEC + 1;
        P = BIT (P_DEC,15);
    END;
END SPEC_J;

PAR_CHECK: PROCEDURE (CHECK) CHAR (4); CHECK ')(SKIP,A);
PUT FILE(SYSPRINT) EDIT ('IN PROC PAR ')(SKIP,A);
DCL CHECK BIT (30);
DCL FINAL CHAR (4);
DCL FLAG FIXED BINARY (15) INITIAL (0);
DCL COUNTER FIXED BINARY (15);
DO COUNTER = 1 TO 30;
    IF SUBSTR (CHECK,COUNTER,1) = '1'B THEN IF FLAG = 1 THEN
        FLAG = 0;
    ELSE FLAG = 1;
END;
IF FLAG = 0 THEN FINAL = 'EVEN';
ELSE FINAL = 'ODD ';
RETURN (FINAL);
END PAR_CHECK;

MEMORY: PROCEDURE (ALPHA) BIT (30);
PUT FILE (SYSPRINT) EDIT ('IN PROC MEMORY ALPHA IS ',ALPHA)
(SKIP,A,F(8));
DCL ALPHA FIXED BINARY (15);
IF ALPHA > 65536 THEN RETURN (MEM1(0));
IF ALPHA > 32767 THEN RETURN (MEM2 (ALPHA - 13516));
ELSE RETURN (MEM1 (ALPHA));
END MEMORY;

FINISH: END SIMUL;

```

```

SIM06100
SIM06110
SIM06120
TIM01910
TIM01920
TIM01930
TIM01940
TIM01950
TIM01960
TIM01970
TIM01980
TIM01990
TIM02000
TIM02010
TIM02020
TIM02030
TIM02040
TIM02050
TIM02060
SIM06130

SIM06140
SIM06150
SIM06160
SIM06170
SIM06180
SIM06190
SIM06200
SIM06210
SIM06220

SIM06240
SIM06250
SIM06260
SIM06270
SIM06280

SIM06290
LES00010
SIM06300
SIM06310
SIM06320
SIM06330
SIM06340
LES00390

```


CP 901 ASSEMBLER LISTING

```

MASTER: PROC OPTIONS (MAIN);
DCL INPUT CHAR (80) VAR;
DCL BUF CHAR (80) VAR;
DCL CODE BIT (30);
DCL LAB FIXED BINARY (15);
DCL MACH (1000) BIT (30);
DCL FORMAT FIXED BINARY (15);
DCL SYSPRINT FILE STREAM PRINT ENV (F(133));
DCL OP CHAR (3);
DCL JDEC FIXED BINARY (15);
DCL J BIT (3) VAR;
DCL J_HAT BIT (4) VAR;
DCL K_HAT BIT (3);
DCL K_HAT BIT (2);
DCL OP CODE BIT (6);
DCL Y BIT (15);
DCL Y_1 BIT (9);
DCL Y_2 BIT (17);
DCL B BIT (3);
DCL SP_77 BIT (6);
DCL WORD (2) CHAR (5) VAR;
DCL WORDM CHAR (5);
DCL (M,N,O) FIXED BIN (15);
DCL CNT FIXED BIN (15) INITIAL (1);
DCL FORM NO ENTRY (FIXED BINARY (15));
DCL VAR (48) LABEL;
DCL TYPE CHAR (144);
DCL TYPE = 'ADD SUB MUL DIV EXP LEQ EQ GEQ GTR NEG';
DCL TYPE = 'NOT AND BOP ONE TWO LOD STODEL STOX ITB FNB FC';
DCL TYPE = 'B C B R S B S C N O P P R O K T N G E T R E T R D V W R S D M P';
DCL TYPE = 'TAB ROW S C R S A V U N S S U P L I T E O F D E L D U P X C H O 4 8';
DCL REV_OCT ENTRY (CHAR(1)) RETURNS (BIT (3));
DCL REV_ENTRY (CHAR(1)) RETURNS (BIT (3));
START: M = 1;
SP_77 = '111111';
DISPLAY ('I AM READY TO WORK') REPLY (BUF);
DO N = 1 TO 2;
WORD (N) = '';
END;
/*
THIS SECTION CHECKS FOR A LABEL INPUT, THE
ASCERTAINS THE LABEL NUMBER AND PUTS THE
PROPER ADDRESS OF THE PRESENT INSTRUCTION INTO *
THE MEMORY LOCATION OF THE LABEL NUMBER.
*/
IF SUBSTR (BUF,1,1) = '5' THEN DO; /* LABELS BEGIN WITH 5 */
LAB = REV_OCT ('00'); SUBSTR (BUF,1,3);
BUF = SUBSTR (BUF,6); /* REMOVES LABEL FROM BUFFER */
K = '000';

```

RAY000020
 RAY000030
 RAY000040
 RAY000050
 RAY000060
 RAY000070
 RAY000080
 RAY000090
 RAY000100
 RAY000110
 RAY000120
 RAY000130
 RAY000140
 RAY000150
 RAY000160
 RAY000170

RAY000180
 RAY000190
 RAY000200
 RAY000210
 RAY000220
 RAY000230
 RAY000240
 RAY000250
 RAY000260
 RAY000270
 RAY000280
 RAY000290
 RAY000300
 RAY000310
 RAY000320
 RAY000330
 RAY000340
 RAY000350
 RAY000360
 RAY000370
 RAY000380
 RAY000390
 RAY000400
 RAY000410
 RAY000420
 RAY000430
 RAY000440
 RAY000450
 RAY000460
 RAY000470


```

RAY000480
RAY000490
RAY000500
RAY000510
RAY000520
RAY000530
RAY000540
RAY000550
RAY000560
RAY000570
RAY000580
RAY000590
RAY000600
RAY000610
RAY000620
RAY000630
RAY000640
RAY000650
RAY000660
RAY000670
RAY000680
RAY000690
RAY000700
RAY000710
RAY000720
RAY000730
RAY000740
RAY000750
RAY000760
RAY000770
RAY000780
RAY000790
RAY000800
RAY000810
RAY000820
RAY000830
RAY000840
RAY000850
RAY000860
RAY000870
RAY000880
RAY000890
RAY000900
RAY000910
RAY000920
RAY000930
RAY000940
RAY000950

J = '000';
B = '000';
Y = BIT (CNT+2,15);
OPCODE = '001001'B;
CALL FORM_NO (1);
K = '011';
OPCODE = '001101'B;
Y = BIT (LAB,15);
J = '000'B;
B = '000'B;
CALL FORM_NO (1);
/* Y HAS LABEL NUMBER */
/* GENERATE MACHINE CODE TO STORE A */
END;
AGAIN: IF SUBSTR (BUF,1,1) ^= ' ' THEN
      WORD (M) || SUBSTR (BUF,1,1);
ELSE M = M + 1;
BUF = SUBSTR (BUF,2);
IF SUBSTR (BUF,1,2) = ' ' THEN GO TO COMPUTE;
GO TO AGAIN;
COMPUTE: OP = WORD (1);
JDEC = TRUNC (INDEX (TYPE,OP) / 3) + 1;
PUT FILE (SYSPRINT) EDIT ('JDEC IS ', JDEC) (SKIP,A,F(6));
GO TO VAR (JDEC);

VAR (1): PUT FILE (SYSPRINT) EDIT ('ADD') (SKIP,A);
OPCODE = '010000'B;
K = '000'B;
J = '000'B;
B = '000'B;
FORMAT = 1;
Y = REV_OCT (WORD (2));
PUT FILE (SYSPRINT) EDIT ('Y AFTER REV OCT IN IS ',Y) (SKIP,A,F(6));
CALL FORM_NO (FORMAT);
GO TO ENDING;

VAR (2): PUT FILE (SYSPRINT) EDIT ('SUB') (SKIP,A);
OPCODE = '010001'B;
K = '000';
J = '000';
B = '000';
FORMAT = 1;
Y = REV_OCT (WORD (2));
PUT FILE (SYSPRINT) EDIT ('Y AFTER REV OCT IN IS ',Y) (SKIP,A,F(6));
CALL FORM_NO (FORMAT);
GO TO ENDING;

VAR (3): PUT FILE (SYSPRINT) EDIT ('MUL') (SKIP,A);
K = '000';
B = '000';

```


RAY000960
RAY000970
RAY000980
RAY000990
RAY01000
RAY01010
RAY01020
RAY01030
RAY01040
RAY01050
RAY01060
RAY01070
RAY01080
RAY01090
RAY01100
RAY01110
RAY01120
RAY01130
RAY01140
RAY01150
RAY01160
RAY01170
RAY01180
RAY01190
RAY01200
RAY01210
RAY01220
RAY01230
RAY01240
RAY01250
RAY01260
RAY01270
RAY01280
RAY01290
RAY01300
RAY01310
RAY01320
RAY01330
RAY01340
RAY01350
RAY01360
RAY01370
RAY01380
RAY01390
RAY01400
RAY01410
RAY01420
RAY01430

```
J = '000';
OPCODE = '001101';
CALL FORM_NO (1);
OPCODE = '010010'; (2);
Y = REV_OCT (WORD (2));
CALL FORM_NO (1);
OPCODE = '000111';
SUBSTR (Y,10,6) = '011000';
CALL FORM_NO (1);
GO TO ENDING;

VAR (4): PUT FILE (SYSPRINT) EDIT ('DIV') (SKIP,A);
K = '000';
J = '000';
B = '000';

OPCODE = '001101';
CALL FORM_NO (1);
Y = '000000000000';
OPCODE = '001001';
CALL FORM_NO (1);
Y = REV_OCT (WORD (2));
OPCODE = '010011';
CALL FORM_NO (1);
SUBSTR (Y,10,6) = '011000';
OPCODE = '000111';
CALL FORM_NO (1);
GO TO ENDING;

VAR (5): PUT FILE (SYSPRINT) EDIT ('EXP') (SKIP,A);
J = '000';
K = '000';
B = '000';
OPCODE = '001101';
CALL FORM_NO (1);
Y = '0000000110010';
CALL FORM_NO (1);
M = REV_OCT (WORD (2));
OPCODE = '010010';
DO WHILE (M ^= 0);
CALL FORM_NO (1);
M = M - 1;
END;
OPCODE = '000111';
SUBSTR (Y,10,6) = '011000';
CALL FORM_NO (1);
GO TO ENDING;
```


RAY01440
RAY01450
RAY01460
RAY01470
RAY01480
RAY01490
RAY01500
RAY01510
RAY01520
RAY01530
RAY01540
RAY01550
RAY01560
RAY01570
RAY01580
RAY01590
RAY01600
RAY01610
RAY01620
RAY01630
RAY01640
RAY01650
RAY01660
RAY01670
RAY01680
RAY01690
RAY01700
RAY01710
RAY01720
RAY01730
RAY01740
RAY01750
RAY01760
RAY01770
RAY01780
RAY01790
RAY01800
RAY01810
RAY01820
RAY01830
RAY01840
RAY01850
RAY01860
RAY01870
RAY01880
RAY01890
RAY01900
RAY01910

```

VAR (6): PUT FILE (SYSPRINT) EDIT ('LES') (SKIP,A);
K = '000'B;
J = '000'B;
B = '000'B;
Y = '0000000000000000'B;
OPCODE = '001101'B;
CALL FORM_NO (1);
Y = '0000000000000001'B;
OPCODE = '010111'B;
CALL FORM_NO (1);
J = '010'B;
Y = REV_OCT (WORD (2));
OPCODE = '000100'B;
CALL FORM_NO (1);
GO TO ENDING;

VAR (7): PUT FILE (SYSPRINT) EDIT ('EQL') (SKIP,A);
GO TO ENDING;

VAR (8): PUT FILE (SYSPRINT) EDIT ('LEQ') (SKIP,A);
J = '110'B;
K = '000'B;
B = '000'B;
Y = REV_OCT (WORD (2));
OPCODE = '000100'B;
CALL FORM_NO (1);
GO TO ENDING;

VAR (9): PUT FILE (SYSPRINT) EDIT ('NEQ') (SKIP,A);
GO TO ENDING;

VAR (10): PUT FILE (SYSPRINT) EDIT ('GEQ') (SKIP,A);
GO TO ENDING;

VAR (11): PUT FILE (SYSPRINT) EDIT ('GTR') (SKIP,A);
J = '111'B;
K = '000'B;
B = '000'B;
Y = REV_OCT (WORD (2));
OPCODE = '000100'B;
CALL FORM_NO (1);
GO TO ENDING;

VAR (12): PUT FILE (SYSPRINT) EDIT ('NEG') (SKIP,A);
GO TO ENDING;
/* SHIFT A TO Q */
/* SUBTRACT 1 FROM Q */
/* COMPARE X WITH Q FOR SKIP */
/* COMPARE X WITH A FOR SKIP */
/* COMPARE X WITH A FRO SKIP */

```


RAY01920
RAY01930
RAY01940
RAY01950
RAY01960
RAY01970
RAY01980
RAY01990
RAY02000
RAY02010
RAY02020

RAY02030
RAY02040
RAY02050

RAY02060
RAY02070
RAY02080
RAY02090
RAY02100
RAY02110
RAY02120
RAY02130
RAY02140
RAY02150
RAY02160
RAY02170
RAY02180
RAY02190
RAY02200
RAY02210
RAY02220

```

VAR (13): PUT FILE (SYSPRINT) EDIT ('NOT') (SKIP,A);
          J = '000'B;
          K = '100'B;
          B = '000'B;
          Y = '0000000000000000'B;
          OPCODE = '001101'B;
          CALL FORM_NC (1);
          GO TO ENDING;
          /* COMPLIMENT A */

VAR (14): PUT FILE (SYSPRINT) EDIT ('AND') (SKIP,A);
          K = '000'B;
          J = '000'B;
          B = '110'B;
          Y = REV_OCT (WORD (2));
          OPCODE = '000100'B;
          CALL FORM_NC (1);
          /* TEST A = X */

          J = '000'B;
          K = '100'B;
          Y = '1111111111111111'B;
          OPCODE = '101001'B;
          CALL FORM_NC (1);
          /* COMPLIMENT */
          GO TO ENDING;

VAR (15): PUT FILE (SYSPRINT) EDIT ('BOR') (SKIP,A);
          B = '000'B;
          J = '000'B;
          K = '100'B;
          Y = '1111111111111111'B;
          OPCODE = '101001'B;
          CALL FORM_NC (1);
          /* COMPLIMENT */
          GO TO ENDING;

VAR (16): PUT FILE (SYSPRINT) EDIT ('ONE') (SKIP,A);
          B = '000'B;
          J = '000'B;
          K = '000'B;
          Y = REV_OCT (WORD (2));
          OPCODE = '001001'B;
          CALL FORM_NC (1);
          GO TO ENDING;
          /* ENTER X INTO A */

VAR (17): PUT FILE (SYSPRINT) EDIT ('TWO') (SKIP,A);
          B = '000';
          J = '000';
          K = '000';
          Y = REV_OCT (WORD (2));
          OPCODE = '001001';

```



```

CALL FORM_NO (1);
Y = REV_OCT (WORD (3));
OPCODE = '001000';
CALL FORM_NO (1);
GO TO ENDING;

VAR (18): PUT FILE (SYSPRINT) EDIT ('LQD') (SKIP,A);
B = '000';
J = '000';
K = '000';
Y = REV_OCT (WORD (2));
OPCODE = '001001';
CALL FORM_NO (1);
GO TO ENDING;

VAR (19): PUT FILE (SYSPRINT) EDIT ('STO') (SKIP,A);
B = '000';
J = '000';
K = '011';
Y = REV_OCT (WORD (2));
OPCODE = '001101';
CALL FORM_NO (1);
GO TO ENDING;

VAR (20): PUT FILE (SYSPRINT) EDIT ('DEL') (SKIP,A);
B = '000'B;
J = '000'B;
K = '000'B;
Y = '0000000000000000'B;
OPCODE = '001001'B;
CALL FORM_NO (1);
/* ZERO OUT A */
GO TO ENDING;

VAR (21): PUT FILE (SYSPRINT) EDIT ('STD') (SKIP,A);
K = '011'B;
Y = REV_OCT (WORD (2));
OPCODE = '001101'B;
CALL FORM_NO (1); /* STORE A IN ADDRESS X */
Y = '0000000000000000'B;
OPCODE = '001001'B;
K = '000'B;
CALL FORM_NO (1); /* ZERO OUT A */
GO TO ENDING;

VAR (22): PUT FILE (SYSPRINT) EDIT ('XIT') (SKIP,A);
B = '000'B;
J = '000'B;
K = '000'B;

```

RAY02230
RAY02240
RAY02250
RAY02260
RAY02270
RAY02280
RAY02290
RAY02300
RAY02310
RAY02320
RAY02330
RAY02340
RAY02350
RAY02360
RAY02370
RAY02380
RAY02390
RAY02400
RAY02410
RAY02420
RAY02430
RAY02440
RAY02450
RAY02460
RAY02470

RAY02480
RAY02490
RAY02500

RAY02510
RAY02520
RAY02530


```

Y = '000000110010000'B; /* BRANCH TO IPL POINT */
OPCODE = '110001'B;
CALL FORM_NO (1); /* JUMP TO 620 */
GO TO ENDING;

VAR (23): PUT FILE (SYSPRINT) EDIT ('BFN') (SKIP,A);
GO TO ENDING;

VAR (24): PUT FILE (SYSPRINT) EDIT ('BFC') (SKIP,A);
GO TO ENDING;

VAR (25): PUT FILE (SYSPRINT) EDIT ('BBC') (SKIP,A);
GO TO ENDING;

VAR (26): PUT FILE (SYSPRINT) EDIT ('BRS') (SKIP,A);
B = '000'B;
J = '001'B;
K = '000'B;
Y = REV OCT (WORD (2));
OPCODE = '110000'B;
CALL FORM_NO (1); /* BRANCH TO X */
GO TO ENDING;

VAR (27): PUT FILE (SYSPRINT) EDIT ('BSC') (SKIP,A);
B = '000'B;
J = '101'B;
K = '000'B;
Y = REV OCT (WORD (2));
OPCODE = '110000'B; /* JUMP TO X */
CALL FORM_NO (1);
GO TO ENDING;

VAR (28): PUT FILE (SYSPRINT) EDIT ('NOP') (SKIP,A);
B = '000'B;
J = '000'B;
K = '000'B;
Y = '0000000000000000'B;
OPCODE = '001010'B; /* ENTER B ZERO WITH ZERO */
CALL FORM_NO (1);
GO TO ENDING;

VAR (29): PUT FILE (SYSPRINT) EDIT ('PRO') (SKIP,A);
GO TO ENDING;

VAR (30): PUT FILE (SYSPRINT) EDIT ('RTN') (SKIP,A);
GO TO ENDING;

VAR (31): PUT FILE (SYSPRINT) EDIT ('GET') (SKIP,A);

```

RAY02540
RAY02550
RAY02560
RAY02570
RAY02580
RAY02590
RAY02600
RAY02610
RAY02620
RAY02630
RAY02640
RAY02650

RAY02660
RAY02670
RAY02680

RAY02690
RAY02700
RAY02710

RAY02720
RAY02730
RAY02740
RAY02750
RAY02760
RAY02770
RAY02780
RAY02790
RAY02800

RAY02810
RAY02820
RAY02830
RAY02840
RAY02850
RAY02860

```

GO TO ENDING;

VAR (32): PUT FILE (SYSPRINT) EDIT ('RET') (SKIP,A);
GO TO ENDING;

VAR (33): PUT FILE (SYSPRINT) EDIT ('RDV') (SKIP,A);
B = '000'B;
J = '000'B;
K = '000'B;
Y = '000000001110010'B;
OPCODE = '001001'B; /*
CALL FORM_NO (1); /* ENTER 162 INTO A */
Y = '00000000001101'B;
OPCODE = '000110'B; /*
CALL FORM_NO (1); /* SHIFT A LEFT 15 BITS */
K = '011'B;
Y = '000000001000010'B;
OPCODE = '001101'B; /*
CALL FORM_NO (1); /* STORE A AT 102 */
J_HAT = '0010'B;
K_HAT = '00'B;
Y = '000000001110010'B;
OPCODE = '111011'B; /*
CALL FORM_NO (1); /* INPUT VAR AT 162 */
K = '000'B;
Y = '000000000000000'B;
OPCODE = '001001'B;
CALL FORM_NO (1); /* SET A = 0 */
Y = '000000001110010'B;
OPCODE = '010100'B; /*
CALL FORM_NO (1); /* SET A = MEM ( 162) */
GO TO ENDING;

VAR (34): PUT FILE (SYSPRINT) EDIT ('WRV') (SKIP,A);
B = '000'B;
J = '000'B;
K = '011'B;
Y = '000000001110010'B;
OPCODE = '001101'B; /*
CALL FORM_NO (1); /* STORE A IN 162 */
K = '000'B;
Y = '000000001110010'B;
OPCODE = '001001'B; /*
CALL FORM_NO (1); /* ENTER 162 INTO A */
Y = '00000000001101'B;
OPCODE = '000110'B; /*
CALL FORM_NO (1); /* SHIFT A LEFT 15 BITS */
K = '011'B;

```

RAY02870
RAY02880
RAY02890


```

Y = '000000001000010'B;
OPCODE = '001101'B; /* STORE A AT 102 */
CALL FORM_NO (1);
K_HAT = '00'B;
J_HAT = '0010'B;
OPCODE = '111011'B;
Y = '000000001110010'B;
CALL FORM_NO (1); /* OUTPUT THE VARIABLE */
GO TO ENDING;

VAR (35): PUT FILE (SYSPRINT) EDIT ('WRS') (SKIP,A);
B = '000'B;
J = '000'B;
K = '000'B;
OPCODE = '001001'B;
Y = REV_OCT (WORD (2)); /* ENTER X INTO A */
CALL FORM_NO (1); /*
Y = '00000000000101'B;
OPCODE = '010000'B; /* ADD 5 TO A */
CALL FORM_NO (1);
Y = '00000000001101'B;
OPCODE = '000110'B; /* SHIFT A LEFT 15 BITS */
CALL FORM_NO (1);
K = '011'B;
Y = '000000001010010'B;
OPCODE = '001101'B;
Y = '000000001000010'B; /* STORE A AT 102 */
CALL FORM_NO (1);
J_HAT = '0010'B;
K_HAT = '00'B;
OPCODE = '111011'B;
Y = '00000000000101'B; /*
CALL FORM_NO (2);
GO TO ENDING;

VAR (36): PUT FILE (SYSPRINT) EDIT ('DMP') (SKIP,A);
GO TO ENDING;

VAR (37): PUT FILE (SYSPRINT) EDIT ('TAB') (SKIP,A);
GO TO ENDING;

VAR (38): PUT FILE (SYSPRINT) EDIT ('ROW') (SKIP,A);
GO TO ENDING;

VAR (39): PUT FILE (SYSPRINT) EDIT ('SCR') (SKIP,A);
GO TO ENDING;

VAR (40): PUT FILE (SYSPRINT) EDIT ('SAV') (SKIP,A);

```

RAY02900
RAY02910
RAY02920

RAY02930
RAY02940
RAY02950
RAY02960
RAY02970
RAY02980
RAY02990
RAY03000
RAY03010
RAY03020
RAY03030

RAY03050
RAY03060
RAY03070


```

GO TO ENDING;
RAY03080
RAY03090
RAY03100
RAY03110
RAY03120
RAY03130
RAY03140
RAY03150
RAY03160
RAY03170
RAY03180
RAY03190
RAY03200
RAY03210
RAY03220

VAR (41): PUT FILE (SYSPRINT) EDIT ('UNS') (SKIP,A);
GO TO ENDING;
RAY03230
RAY03240
RAY03250
RAY03260
RAY03270
RAY03280
RAY03290
RAY03300
RAY03310
RAY03320
RAY03330
RAY03340
RAY03350
RAY03360
RAY03370
RAY03380
RAY03390
RAY03400
RAY03410
RAY03420
RAY03430
RAY03440
RAY03450
RAY03460
RAY03470
RAY03480
RAY03490
RAY03500
RAY03510
RAY03520
RAY03530

VAR (42): PUT FILE (SYSPRINT) EDIT ('SUP') (SKIP,A);
GO TO ENDING;

VAR (43): PUT FILE (SYSPRINT) EDIT ('LIT') (SKIP,A);
B = '000'B;
J = '000'B;
K = '000'B;
Y = REV_OCT (WORD (2));
OPCODE = '001001'B;
CALL FORM_NO (1); /* SET A = X */
OPCODE = '001000'B; /* SET Q = X */
CALL FORM_NO (1); /* SET Q = X */
GO TO ENDING;

VAR (44): PUT FILE (SYSPRINT) EDIT ('EOF') (SKIP,A);
J = '100';
K = '000';
B = '000';
Y = '0000000000000000';
OPCODE = '110001';
CALL FORM_NO (1);
GO TO ENDING;

VAR (45): PUT FILE (SYSPRINT) EDIT ('DEL') (SKIP,A);
B = '000'B;
J = '000'B;
K = '000'B;
Y = '0000000000000000'B;
OPCODE = '001001'B;
CALL FORM_NO (1);
GO TO ENDING; /* ENTER A 0 INTO A */

VAR (46): PUT FILE (SYSPRINT) EDIT ('DUP') (SKIP,A);
B = '000'B;
J = '000'B;
K = '000'B;
Y = '0000000000000000'B;
OPCODE = '001101'B;
CALL FORM_NO (1);
GO TO ENDING; /* SET Q = A */

VAR (47): PUT FILE (SYSPRINT) EDIT ('XCH') (SKIP,A);
B = '000'B;

```


RAY03540
RAY03550
RAY03560
RAY03570
RAY03580
RAY03590
RAY03600
RAY03610
RAY03620
RAY03630
RAY03640
RAY03780
RAY03790
RAY03800
RAY03810
RAY03820

```
J = '000'B;
K = '000'B;
Y = '000000000011000'B;
OPCODE = '000111'B;
CALL FORM_NO (1);
GO TO ENDING;
/* SHIFT AQ CIRCULARLY 30 BITS */
```

```
VAR (48): PUT FILE (SYSPRINT) EDIT ('48') (SKIP,A);
GO TO ENDING;
```

```
ENDING: DO 0 = 1 TO CNT - 1;
PUT FILE (SYSPRINT) EDIT ('MACHINE', 0, ' IS ', MACH (0))
(SKIP,A,F(6),A,B(30));
```

```
END;
GO TO START;
```

```
/* PROCEDURE FORM_NO ASSEMBLES THE VARIOUS FIELDS OF
THE INSTRUCTION WORDS INTO THE PROPER FORMAT AS
DESIGNATED BY THE PARAMETER FORM BY THE TYPE OF
INSTRUCTION .
*/
```

RAY03650
RAY03660
RAY03670
RAY03680
RAY03690
RAY03700
RAY03710
RAY03720
RAY03730
RAY03750

```
FORM_NO: PROCEDURE (FORM);
DCL FORM FIXED BINARY (15);
IF FORM = 1 THEN DO;
PUT FILE (SYSPRINT) EDIT ('CNT IN FORMAT ',CNT) (SKIP,A,F(6));
PUT FILE (SYSPRINT) EDIT ('OPCODE IN FORMAT IS ',J) (SKIP,A,B(3));
PUT FILE (SYSPRINT) EDIT ('J IN FORMAT ONE IS ',J) (SKIP,A,B(3));
PUT FILE (SYSPRINT) EDIT ('K IN FORMAT ONE IS ',K) (SKIP,A,B(3));
PUT FILE (SYSPRINT) EDIT ('B IN FORMAT ONE IS ',B) (SKIP,A,B(3));
MACH (CNT) = OPCODE || J || K || B || Y;
END;
```

```
IF FORM = 2 THEN
MACH (CNT) = OPCODE || J_HAT || K_HAT || B || Y;
IF FORM = 3 THEN
MACH (CNT) = SP_77 || OPCODE || B || Y;
IF FORM = 4 THEN
MACH (CNT) = SP_77 || OPCODE || B || J_HAT || K_HAT || Y_1;
IF FORM = 5 THEN
MACH (CNT) = SP_77 || OPCODE || '0'B || Y_2;
CNT = CNT + 1;
END FORM_NO;
```

RAY03740
RAY03760
RAY03770
RAY03830

```
/* PROCEDURE REV_OCT ACCEPTS THE PARAMETER OF A FIVE
DIGIT CHARACTER STRING AND RETURNS THE OCTAL
BIT REPRESENTATION FOR THE STRING.
*/
```


RAY03840
RAY03850
RAY03860
RAY03870
RAY03880
RAY03890
RAY03900
RAY03910
RAY03920
RAY03930

RAY03940
RAY03950
RAY03960
RAY03970
RAY03980
RAY03990
RAY04000
RAY04010
RAY04020
RAY04030
RAY04040
RAY04050
RAY04060
RAY04070

*/

```

REV_OCT: PROCEDURE (VALUE) BIT (15);
  DCL VALUE CHAR (5);
  DCL MM FIXED BINARY (15);
  DCL RET BIT (15) VAR INITIAL ('');
  DO MM = 1 TO 5;
    RET = RET || REV (SUBSTR (VALUE,MM,1));
  END;
  RETURN (RET);
END REV_OCT;

```

/* PROCEDURE REV RETURNS THE OCTAL BIT REPRESENTATION
OF THE SINGLE NUMERIC CHARACTER INPUT.

```

REV: PROCEDURE (ONECHAR) BIT (3);
  DCL ONECHAR CHAR (1);
  DCL THING BIT (3);
  IF ONECHAR = '0' THEN RETURN ('000'B);
  IF ONECHAR = '1' THEN RETURN ('001'B);
  IF ONECHAR = '2' THEN RETURN ('010'B);
  IF ONECHAR = '3' THEN RETURN ('011'B);
  IF ONECHAR = '4' THEN RETURN ('100'B);
  IF ONECHAR = '5' THEN RETURN ('101'B);
  IF ONECHAR = '6' THEN RETURN ('110'B);
  ELSE RETURN ('111'B);
END REV;
END MASTER;

```


BIBLIOGRAPHY

1. Sperry Rand Corporation, UNIVAC 1830-A Computer Technical Description.
2. Kildall, G. A., The Balgol-2 Programming System, Naval Postgraduate School, 1970.
3. McKeeman, W. M., Horning, J. J., and Wortman, D. B., A Compiler Generator, Prentice-Hall, Inc., 1970.
4. Orlicky, J., The Successful Computer System, McGraw Hill Book Company, 1969.
5. IBM System/360 PL/I Reference Manual, International Business Machine Company, 1968.
6. P-3C ORION Flight Crew Training, Volume I, Department of the Navy 12ND NASMF P1543 (VP-31), 1970.
7. P-3C ORION Flight Crew Training, Volume II, Department of the Navy 12ND NASMF P1543 (VP-31), 1970.
8. CP 901 ASQ-114 Computer Word Formats, Department of the Navy P-208-015-0, 1969.

INITIAL DISTRIBUTION LIST

	No. Copies
1. Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2. Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2
3. Asst Professor G. H. Syms Department of Mathematics Naval Postgraduate School Monterey, California 93940	1
4. Lt. G. A. Kildall, USN Department of Mathematics Naval Postgraduate School Monterey, California 93940	1
5. Lt. Dennis E. Ray, USN 246 Harlem Road Pasadena, Maryland 21122	1
6. LCDR E. A. Singer, USN ASW Systems Project Office Code ASW 251 Navy Department Washington, D. C. 20360	1
7. Commanding Officer VP-31 Attn: LCDR Grant Fuller, USN NAS Moffett Field Mountain View, California	3

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

ORIGINATING ACTIVITY (Corporate author)

Naval Postgraduate School
Monterey, California 93940

2a. REPORT SECURITY CLASSIFICATION

Unclassified

2b. GROUP

REPORT TITLE

The Design, Development and Translation of General Purpose Software for the
P3C Aircraft's Digital Computer

DESCRIPTIVE NOTES (Type of report and, inclusive dates)

Master's Thesis; December 1971

AUTHOR(S) (First name, middle initial, last name)

Dennis Edward Ray

REPORT DATE

December 1971

7a. TOTAL NO. OF PAGES

91

7b. NO. OF REFS

8

CONTRACT OR GRANT NO.

PROJECT NO.

9a. ORIGINATOR'S REPORT NUMBER(S)

9b. OTHER REPORT NO(S) (Any other numbers that may be assigned
this report)

DISTRIBUTION STATEMENT

Approved for public release; distribution unlimited.

SUPPLEMENTARY NOTES

12. SPONSORING MILITARY ACTIVITY

Naval Postgraduate School
Monterey, California

ABSTRACT

The Navy has incorporated a modified UNIVAC 1830-A (CP 901 or ASQ/114) "mini" digital computer into its P3C aircraft. This ASQ/114 computer system is presently used only during aircraft testing and flying. In the near future, fifty or more of these digital systems will be operational and will sit virtually idle about 40% of the time. Hence, this project was undertaken to improve the computer utilization, and to provide the individual squadrons with an administrative computer capability.

Six specific tasks leading to the implementation of a general purpose operating system have been undertaken: a feasibility study, development of a CP 901 translator, design and development of an assembler, design study of the bootstrapping technique, design of a FORTRAN compiler, and the design of a control operating system.

The documentation of these six tasks is intended to aid in the development of the final system.

KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
P3C General Purpose Software Digital Computer						



Thesis
R247
c.1

Ray

133475

The design, develop-
ment and translation
of general purpose
software for the P3C
aircraft's digital
computer.

23 MAY 73
7 NOV 73
31 JUL 76
15 DEC 86

21793
22221
- 24292
31315

Thesis
R247
c.1

Ray

133475

The design, develop-
ment and translation
of general purpose
software for the P3C
aircraft's digital
computer.

thesR247

The design, development and translation



3 2768 002 05322 5

DUDLEY KNOX LIBRARY